ABC INFORMATIQUE

N°87

COURS D'INFORMATIQUE PRATIQUE ET FAMILIALE

Le compatible Tandy 1000

Cadrage sur le C64

Jeu de piste et intelligence

Des livres sur votre micro

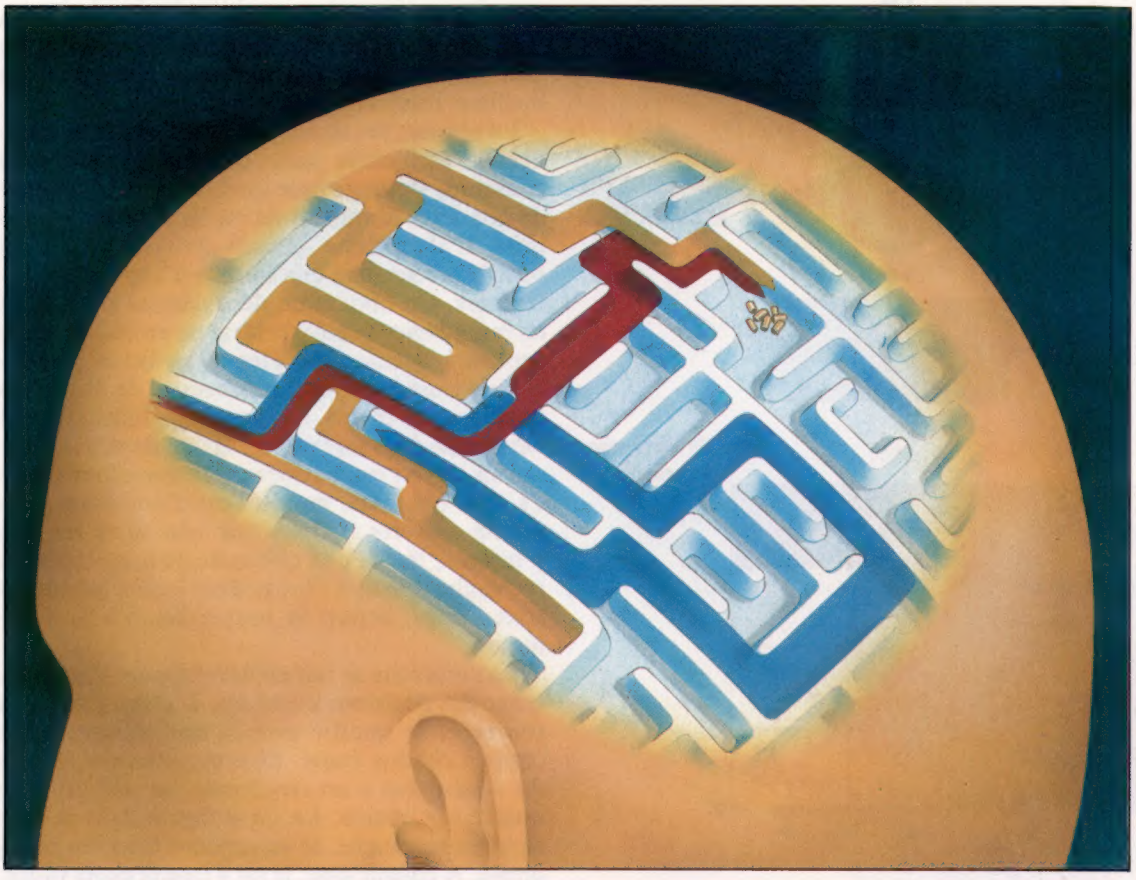
EDITIONS ATLAS

M 6062-87-12,50 F 90 FB - 3,80 FS - $ 1.95

Page manquante
(publicité et colophon)

Jeu de pistes

A l'aide de l'énigme classique du « rat dans le labyrinthe », voyons quelques principes fondamentaux des techniques d'intelligence artificielle permettant de résoudre des problèmes.



Courses de rats
Ce dessin montre les trois cheminements possibles dans le labyrinthe : stratégie systématique et stratégie heuristique. Dans le cas présent, l'approche heuristique est la plus rapide, même s'il est parfaitement possible de dessiner un labyrinthe où ce ne serait pas le cas. Dans la pratique, c'est la combinaison des méthodes heuristique et exhaustive qui se révèle la plus efficace. (Cl. Steve Cross.)

Imaginez trois rats abandonnés au milieu d'un labyrinthe dans lequel de la nourriture a été déposée : un des rats erre quelques minutes et s'effondre, endormi (l'expérimentateur, cruel, avait mélangé de l'alcool à l'eau que l'animal devait boire).

Le deuxième est plus méthodique (!) : il suit de sa patte gauche le mur, et ne prend ainsi que des virages à gauche, revenant sur ses pas lorsqu'il rencontre un cul-de-sac. Il finit par arriver à destination, mais trop tard : le troisième rat a déjà tout mangé. C'est que le troisième rat a de l'odorat...

Ce dernier s'est souvent arrêté dans sa course pour renifler l'air, et a choisi le chemin qu'il estimait le plus rapproché de l'odeur de nourriture. Il faut néanmoins convenir que de se fier à l'odorat nous semble l'attitude la plus intelligente pour un rat.

La recherche d'une solution à un problème occupe une place essentielle en intelligence artificielle. Qu'il s'agisse d'un trésor que vous recherchez en scaphandre par 60 m de fond dans les mers des Caraïbes ou de jeux d'énigmes, vous cherchez quelque chose. La résolution d'un problème est ainsi identifiée à la recherche de ses solutions.

Ce rapprochement s'est révélé très fructueux. En effet, de nombreux problèmes différents peuvent être ainsi résolus. Nos trois rats illustrent trois stratégies bien distinctes :

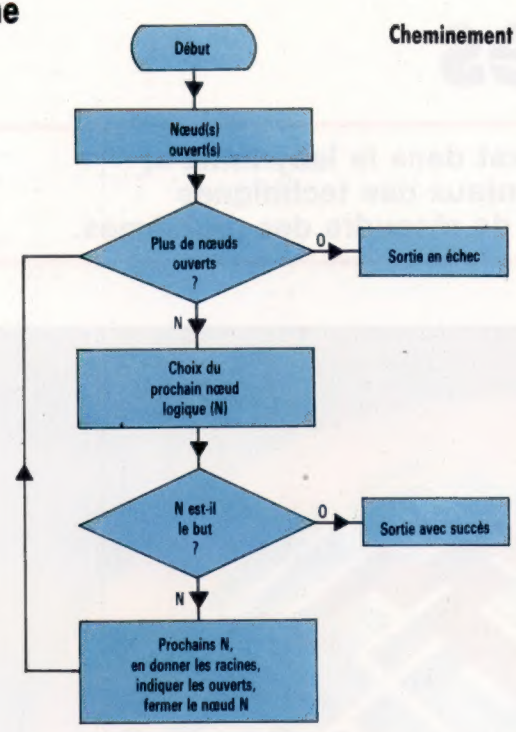
- Stratégie aléatoire (« la démarche du matelot ivre »).
- Recherche exhaustive (énumération systématique).
- Approche heuristique (exploration guidée).

La troisième est la plus intelligente, du fait qu'elle demande moins d'efforts pour parvenir au but. Mais toutes les méthodes heuristiques supposent un moyen de renseignement, de connaissance. Il s'agit pour le rat d'apprécier le chemin le meilleur qui le rapproche du but (dans le cas du troisième rat, par l'odorat). Sans connaissances, pas d'intelligence : il ne reste que l'énumération systématique. Trouver son chemin dans un labyrinthe est un problème-type de recherche. Il faut

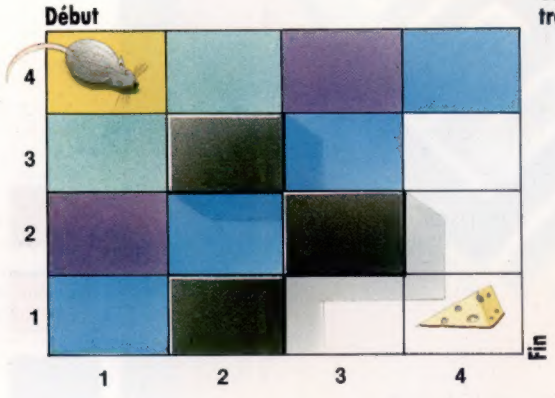
Questions de recherche

La plupart des recherches en intelligence artificielle font intervenir des arborescences représentant la disposition des labyrinthes à parcourir. L'arbre donné ici prend racine à l'entrée du labyrinthe. Il s'épanouit avec la multiplication des embranchements possibles. Remarquez que les déplacements en diagonale ne sont pas permis. L'organigramme représente l'algorithme de base de l'arborescence du labyrinthe.
(Cl. Kevin Jones.)

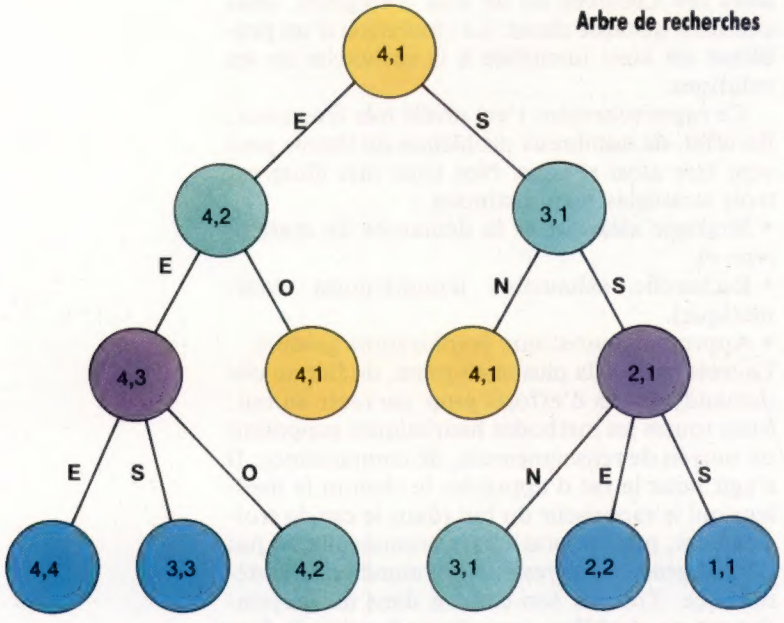
Cheminement



Un labyrinthe très simple



Arbre de recherches



néanmoins savoir que tous les problèmes ne sont pas d'ordre spatial. L'intégration d'une expression symbolique mathématique peut être également traitée par la même méthodologie. Il existe deux critères d'appréciation d'une méthode de recherche :

- Le temps mis pour trouver le cheminement.
- L'efficacité de ce parcours logique (son économie).

De manière idéale, nous recherchons la méthode qui aboutit à la solution optimale en un minimum de temps. En pratique, cependant, nous devons souvent adopter un compromis en prenant soit une méthode rapide à solution non optimale, soit une approche lente pour un meilleur résultat.

La plupart des stratégies de recherche en intelligence artificielle sont fondées sur un plan commun, véritable base modulable. Cela permet de pondérer librement tel ou tel critère d'exécution. L'algorithme général des cheminements possibles regroupe une famille d'approches différentes. On obtient un déroulement particulier en remplissant le pavé central ou en indiquant le prochain carrefour vers lequel se diriger (le prochain « nœud » ou embranchement de directions possibles, ou choix). Les nœuds logiques peuvent ainsi avoir deux statuts : ouvert ou fermé. Un carrefour ouvert est à explorer, un carrefour fermé l'a déjà été d'une manière ou d'une autre. Les nœuds ouverts figurent sur une sorte de liste d'attente ; la clef de l'efficacité de la méthode de recherche tenant à l'ordre dans lequel les nœuds sont traités, depuis la source jusqu'à la destination.

La recherche se fait en développant l'arborescence des décisions possibles. L'arbre part de la source et se ramifie avec la multiplication des choix à chaque étape. Le labyrinthe très simple du « rat-robot » est représenté par une grille de quatre sur quatre. Le rat y figure dans la case supérieure gauche, la nourriture étant en bas à droite. A chaque case, le rat a quatre choix (nord, est, sud et ouest). Certains peuvent néanmoins être interdits. Si nous ouvrons dès le début toutes les portes, nous obtenons la structure arborescente de la représentation des recherches.

De sa position initiale en (4,1), le rat ne peut aller qu'en (4,2) et (3,1). En (4,2), il peut aller à l'est ou à l'ouest. De (3,1), il peut se déplacer au nord ou au sud. Si le rat va à l'est et ensuite à l'ouest (ou bien au sud et ensuite au nord), il revient à sa position de départ. Ce n'est pas un résultat bien brillant, mais il lui permet d'apprendre que ce genre de carré est susceptible de se reproduire en différents points du labyrinthe. Ces carrés indiquent en outre les divers chemins d'accès.

Une des méthodes systématiques d'exploration est horizontale. Chaque carrefour est exploré par ordre de proximité, depuis la racine de l'arbre, le début du labyrinthe. Ainsi, tout déplacement de N mouvements sera essayé avant ceux de N+1. Il s'agit de trouver le plus court chemin, et pourvu que chaque mouvement soit de longueur égale aux autres, c'est la méthode de moindre

coût. Cela dit, le but sera long à atteindre. Au fur et à mesure que l'on s'élève dans l'arbre, les possibilités à explorer augmentent, et le temps d'exploration s'accroît exponentiellement.

L'amélioration de cette méthode horizontale suppose un moyen de renseignement sur la proximité du but. Nous avons donc besoin d'une approche heuristique. Prenons l'exemple suivant : à Manhattan, à New York, toutes les rues et avenues se croisent, disons, à angle droit. Le sachant, on peut apprécier le nombre de « blocs » de maisons à franchir par une démarche nord ou sud, et est ou ouest. Un rat intelligent peut ainsi apprécier la distance qui le sépare de la cible.

En sachant à quel moment on s'approche du but, on peut accélérer grandement les recherches. Le rat du début suivait la stratégie « au plus prêt ». C'est la stratégie du grimpeur qui se guide sur le haut d'une colline. En cas de brouillard, en effet, on se repère en grimpant sans cesse. C'est une méthode plus rapide, mais elle ne va pas nécessairement par le plus court chemin.

Une meilleure stratégie combine les deux approches. Il s'agit de l'algorithme dit « A* », qui suit la formule suivante : DH + DP (Distance Heuristique + Distance Parcourue). Plus l'estimation de DH est fine, plus la recherche est efficace. Lorsque DH n'est pas précise, il convient de le sous-estimer.

Dans notre exemple de programme, nous utilisons un seul canevas pour les trois méthodes.

Méthode horizontale — plus petit DP.

Le grimpeur — plus petit DH.

Algorithme A* — plus petit DH + DP.

Programme de recherches dans le labyrinthe sur C64

```
1000 REM ****************************************
1010 REM ** listage 2.1 :                        **
1020 REM ** Programme de Recherches Labyrinthe  **
1030 REM ****************************************
1040 FOR I=1 TO 25: DW$=DW$+CHR$(17):NEXT I
1050 MH=17: MW=25: REM Largeur et Profondeur du Labyrinthe
1060 SI=256 : REM limites de l'arbre
1070 WA=1 : RR=2: FO=3: DN=4: BL=5
1080 W1=1 : REM limites de l'arbre
1090 W2=2 : REM largeur pour HD
1100 DIM M(MW+1,MJ+1) : REM le labyrinthe
1110 DIM C$(5) : REM caractères labyrinthe
1120 DIM P(SI),S(SI),N(SI),H(SI)
1130 REM P=Chemin,s=étapes, N=Noeuds, Heuristique=distance
1140
1150 REM -- Rat dans labyrinthe :
1160 GOSUB 1360 : REM faire le labyrinthe
1170 NC=0 : REM nombre de noeuds essayés
1180 K=0 : REM compteur
1190 GOSUB 1660 : REM supprimer tous les chemins
1200 N(1)=2+MW+2 : REM 1er noeud ouvert
1210 S(1)=0: H(1)=FR-1 + (FC-1)
1220 P(1)=0: REM sans racine
1230 REM -- Boucle principale :
1240 REM **** boucle principale ****
1250 GOSUB 1770 : REM noeud suivant
1260 NC=NC+1
1270 P=X:Q=Y:GOSUB
1280 GOSUB 1880 : REM suivants
1290 IF (FR<>SR OR FC<>SC) AND NC<300 THEN 1240
1300 P=X:Q=Y:GOSUB
1310 IF FR=SR AND FC=SC THEN GOSUB 2290 : REM
retracer les étapes
1320 IF NC>300 THEN PRINT "raté"
1330 PRINT NC," noeuds essayés "
1340 END
1350 :
1360 REM -- routine de création de labyrinthe :
1370 FOR P=1 TO MH+1
1380 FOR R=1 TO MW+1
1390 IF RND(1) < .020 THEN M(P,R)=WA ELSE M(P,R)=
BL
1400 IF P=3 OR R=3 THEN M(P,R)=BL
1410 IF P=1 OR R=1 THEN M(P,R)=WA
1420 IF P>MH OR R>MW THEN M(P,R)=WA
1430 NEXT: NEXT
1440 FR=2+INT(RND(1)*(MH-1)): REM avancement horizontal
1450 FC=4+INT(RND(1)*MW-3): REM avancement vertical
1460 M(FR,FC)=FO
1470 M(2,2)=RR : REM rat-robot
1480 C$(BL)=" "
C$(WA)=CHR$(102)
1500 C$(DN)=" "
1510 C$(RR)="R"
1520 C$(FO)="+"
1530 GOSUB 1560 : REM affichez-le
1540 RETURN
1550 :
1560 REM -- routine d'affichage du labyrinthe
PRINT CHR$(147)
1580 FOR R=1 TO MH+1
1590 FOR C=1 TO MW+1
1600 P=X:Q=Y:GOSUB
1610 NEXT
1620 PRINT
1630 NEXT
1640 RETURN
1650 :
1660 REM -- routine d'effacage de l'arbre :
1670 DD=9999 : REM fin
1680 FOR Q=1 TO SI
1690 P(Q)=0
1700 S(Q)=DD
1710 N(Q)=0
1720 H(Q)=DD
1730 NEXT
1740 NN=2 : REM Prochain noeud disponible
1750 RETURN
1760 :
1770 REM -- Passer au meilleur noeud SI
1780 S=1: BN=DD
1790 FOR I=1 TO SI
1800 V=S(I)*W1 + ABS(H(I))*W2
1810 IF V<BN AND H(I)>=0 THEN S=I: BN=V
1820 NEXT
1830 IF S=1 THEN P=X:Q=Y:GOSUB
1840 SR=INT(N(S)/MW)
1850 SC=N(S)-MW*SR
1860 RETURN
1870 :
1880 REM -- routine de création des points noeudaux suivants
1890 IF H(S)=0 THEN RETURN : REM fait
1900 REM -- Nord
1910 Y=SR-1 : X=SC
1920 IF Y>1 THEN GOSUB 2090
1930 REM -- Est :
1940 Y=SR : X=SC+1
1950 IF X<=MW THEN GOSUB 2090
1960 REM -- Sud
1970 Y=SR+1 : X=SC
1980 IF Y<=MH THEN GOSUB2090
1990 REM -- Ouest :
2000 Y=SR : X=SC-1
2010 IF X>1 THEN GOSUB 2090
2020 REM -- fermer aussi le noeud S :
2030 H(S) = -H(S)
2040 IF H(S)>0 THEN PRINT "Ueh!"
2050 P=X:Q=Y:GOSUB
2060 M(SR,SC)=DN
2070 REM effacer les positions à l'écran
2080 :
2090 REM -- Routine d'ouverture de 1 noeud
2100 IF M(Y,X)=DN THEN RETURN
2110 IF M(Y=X)=WA THEN RETURN
2120 REM -- Trouver d'abord une position libre :
2130 NX=0
2140 REM ** boucle Trouver position
2150 IF S(NN)<>DD THEN NX=NX+1: NN=NN+1
2160 IF NN>SI THEN NN=1
2170 IF NX>SI THEN PRINT "Plein ! "!
2180 IFS(NN)<>DD THEN 2140
2190 REM -- L'ouvrir maintenant : STOP
2200 XY=X + Y*MW
2210 N(NN)=XY
2220 P(NN)=S
2230 S(NN)=S(S)+1
2240 H(NN)=ABS(Y-FR) + ABS(X-FC)
2250 P=X:Q=Y:GOSUB
2260 REM l'affiche à l'écran
2270 RETURN
2280 :
2290 REM -- routine retracant le cheminement :
2300 ST=S(S)
2310 FOR Q=1 TO 10000:NEXT Q:GOSUB1560
2320 P=X:Q=Y:GOSUB
2330 REM ** AFFICHER CHEMIN **
2340 S=P(S) : REM noeud apparenté
2350 XY=N(S) : REM coordonnées
2360 Y=INT(XY/MW)
2370 X=XY-Y*MW
2380 M(Y,X)=RR : REM traces du rat !
2390 P=X:Q=Y:GOSUB
2400 IF S>0 THEN 2330
2410 P=X:Q=Y:GOSUB
2420 P=X:Q=Y:GOSUB
2430 PRINT NC;" fermeture des noeuds "
2440 RETURN
```

Lignes pour labyrinthe
Le labyrinthe est représenté sous la forme d'un tableau à deux dimensions M(,), et les structures de données pour l'arbre de recherche figurent aux tableaux P(), S(), N() et H(). Le programme comprend les facteurs « coût-présent » et « coût-estimé ». Ils peuvent être modifiés en changeant les valeurs W1 et W2 aux lignes 1080 et 1090. Le programme est listé pour suivre la stratégie du grimpeur, mais vous pouvez le modifier. La méthode heuristique utilisée est fondée sur le principe dit « des distances à Manhattan », très précis ici. De la sorte, un affinement dans le sens de la méthode heuristique par W2>W1 accélère les recherches.

Variantes de basic

Pour le Spectrum, effectuez les modifications suivantes : Remplacez toutes les occurrences de :

TAB (X,Y) :

par

A X,Y;

et en ligne 1490 :

1490 LET C$(WA)=CHR$(143)

Supprimer la ligne 1040.

	(1)	(2)	(3)
T$()	Perles	Statues	Épices
V1()	2 pièces d'or	2 pièces d'or	1 pièce d'or
V2()	?	?	?
EQ(,) 1 sac de sel	0,5	0,5	1
1 ballot de tissu	5	5	10
1 joyau	3	3	6
1 couteau	2	2	4

Description des marchandises offertes par le chef.

Prix des marchandises quand le bateau quitte le port.

Prix des marchandises quand le bateau revient.

Rapports sur les marchandises à échanger.

Le Nouveau Monde

Votre navire vient d'aborder le Nouveau Monde. Il est possible maintenant de commercer avec les indigènes, pourvu que vous n'entriez pas en guerre avec eux...

Nous en sommes maintenant au stade ultime du jeu : le voyage lui-même prend fin avec votre arrivée dans le Nouveau Monde, et vous allez désormais vous efforcer d'échanger avec les natifs les marchandises que vous avez amenées. Vous aurez ainsi l'occasion de faire des profits lors de votre retour en Europe, en y revendant ce que vous avez acquis. Mais vous n'avez jamais rencontré ceux qui vivent dans ce pays inconnu, et ils peuvent se révéler amicaux ou hostiles... Il est donc très important de ne rien faire qui puisse créer des tensions entre eux et votre équipage.

Une fois que la boucle consacrée au voyage prend fin, la ligne 891 appelle le sous-programme de la ligne 10000 qui gère votre arrivée dans le Nouveau Monde. Alors que votre navire s'approche du rivage, plusieurs groupes d'indigènes armés font leur apparition afin de voir ce qui se passe. Vous devez maintenant prendre une décision très importante : ne pas ouvrir le feu (mais le navire risque d'être attaqué), ou tirer (mais cela peut déclencher une véritable guerre).

La ligne 10015 analyse le second élément du tableau des provisions, A0(2), qui indique combien de fusils il y a à bord. S'il s'en trouve effectivement, vous devrez faire un choix. La ligne 15022 attend une réponse : si vous tapez « oui », la poudre parlera et de nombreux indigènes seront tués. Mais les survivants risquent alors de revenir la nuit et d'incendier le navire — ce qui, vous vous en doutez, mettra fin à la partie... Il est évident qu'une agression injustifiée serait une très mauvaise tactique. C'est bien pourquoi la ligne 10044 a été insérée dans le programme; elle vous empêche de continuer, si par malheur vous avez choisi la « politique de la canonnière ».

Si vous avez choisi de ne pas tirer, il ne se passera rien de dramatique, et la ligne 10026 vous envoie ligne 10050. Les indigènes montent à bord pacifiquement, et se montrent très amicaux avec les membres de l'équipage, qu'ils emmènent ensuite dans leur village pour rencontrer leur chef. Celui-là a déjà rencontré des arrivants de l'Ancien Monde, et il est tout à fait bien disposé envers vous. Il vous offre de quoi vous nourrir

et vous reposer, et les échanges (gérés par un module séparé) peuvent commencer le lendemain. Nous aborderons cet aspect du jeu dans le prochain article. Mais nous devons d'ores et déjà procéder à une mise en place préalable.

Le prix des perles, des épices et des statues avait été fixé lorsque le navire avait quitté le port, mais il a changé entre-temps.

Pour faciliter les échanges, il faut donc créer plusieurs tableaux nouveaux. La ligne 60 DIMensionne le tableau TS(), qui gère les descriptions des trois types de marchandises offertes par le chef : perles, statues, épices. La ligne 61 DIMensionne le tableau V1() contenant les valeurs marchandes de ces objets au moment où le bateau a entrepris son voyage.

Les perles et les statues se vendaient alors deux pièces d'or chacune, et les épices valaient une pièce d'or le gramme. La ligne 62, à son tour, DIMensionne le tableau V2, qui gère le prix des marchandises à l'issue du voyage de retour. Mais il y a toujours des fluctuations, et un élément aléatoire influera sur la valeur finale. La ligne 62 fixera donc à deux pièces d'or, ou deux et demie, l'élément V2(1), qui représente le prix des perles. Celui des statues est mis à une, deux ou trois pièces d'or, et les épices à deux ou deux pièces d'or et demie par gramme, et ce, de la même façon.

Les échanges proprement dits ne se font pas sous une forme monétaire; les marchandises sont échangées selon des taux fixes — un lot contre un autre. Le chef propose au capitaine des lots de perles, de statues et d'épices correspondant à ce que son interlocuteur lui propose.

Les lignes 64 à 68 DIMensionnent un tableau à deux dimensions, et lui assignent des valeurs. Les taux d'échange sont fixés à chaque fois. Le premier indice correspond ainsi aux quatre articles que vous avez à proposer : sel, tissu, couteaux et bijoux. Le second indice gère ce que les indigènes vous proposent : perles, statues et épices. Les taux d'échange sont fixés par trois lignes du programme.

La ligne 64 détermine ce taux pour les sacs de sel (premier élément du premier indice). Même chose pour les perles, premier élément du second indice : EQ(1,1) se voit affecter une valeur qui donne le nombre de perles par sac de sel. Le mettre à 0.5 fixe ainsi le taux d'échange à une demi-perle pour un sac de sel. De même, EQ(1,2) reçoit une valeur correspondant à l'intersection du premier élément du premier indice et du second élément du second indice — le sel et les statues. EQ(1,2) étant mis à 0.5, le taux d'échange est d'une statue pour deux sacs de sel. EQ(1,3) fixe la valeur d'un gramme d'épice — troisième élément du second indice — à un sac de sel.

Le taux d'échange du second élément du premier indice — les ballots de tissu — est déterminé ligne 66; chaque ballot vaut cinq perles, cinq statues ou 10 g d'épices. La ligne 67 prend en compte les couteaux — troisième élément —, dont chacun peut être échangé contre trois perles, trois statues ou 6 g d'épices. Les joyaux — quatrième élément —, sont également gérés par la même ligne : deux perles, deux statues ou 4 g d'épices.

La ligne 68 DIMensionne le tableau AO(3) qui gère les quantités de perles, de statues et d'épices acquises durant les échanges. Il ne faut pas le confondre avec le tableau OA, consacré aux marchandises achetées avant le début du voyage.

Les valeurs définies dans ce module seront mises en œuvre lors des échanges, selon des modalités que nous étudierons plus particulièrement en détail prochainement.

Module onze : l'Arrivée

DIMension tableaux échange

```
60 DIMT$(3):T$(1)="PERLES":T$(2)="STATUES":T$
(3)="EPICES"
61 DIMV1(3):V1(1)=2:V1(2)=2:V1(3)=1
62 DIMV2(3):V2(1)=2+(INT(RND(1)*1)/2):V2(2)=2+
(INT(RND(1)*3)-1)
63 V2(3)=2+(INT(RND(1)*1)/2)
64 DIMEQ(4,3)
65 EQ(1,1)=0.5:EQ(1,2)=0.5:EQ(1,3)=1
66 EQ(2,1)=5:EQ(2,2)=5:EQ(2,3)=10
67 EQ(3,1)=3:EQ(3,2)=3:EQ(3,3)=6
68 EQ(4,1)=2:EQ(4,2)=2:EQ(4,3)=4
69 DIMAO(3)
```

Additions au programme principal

```
890 REM ARRIVEE NOUVEAU MONDE
891 GOSUB10000
```

S/P Arrivée

```
10000 REM ARRIVEE NOUVEAU MONDE
10001 PRINTCHR$(147):GOSUB9200
10005 S$="VOUS VOILA DANS LE NOUVEAU MONDE":GOSUB9100
10006 PRINT:GOSUB9200
10007 S$="COMME VOUS APPROCHEZ DU RIVAGE":GOSUB9100
10009 S$="DES INDIGENES FONT LEUR APPARITION":GOSUB91
00
10010 PRINT:GOSUB9200
10015 IFOA(2)=0THEN10050
10017 S$="ILS SONT ARMES ET ONT L'AIR FEROCE !":GOSUB
9100
10018 PRINT:GOSUB9200
10020 S$="DECIDEZ-VOUS D'OUVRIR LE FEU ? (O/N)":GOSUB9100
10022 INPUTI$:I$=LEFT$(I$,1)
10024 IFI$<>"N"ANDI$<>"O"THEN10022
10026 IFI$="N"THEN10050
10028 PRINT:GOSUB9200
10030 S$="PLUSIEURS SAUVAGES SONT TUES":GOSUB9100
10032 S$="MAIS D'AUTRES REVIENNENT DE NUIT":GOSUB9100
10034 S$="ET PAR SURPRISE":GOSUB9100
10036 S$="INCENDIENT VOTRE BATEAU !!!!":GOSUB9100
10038 PRINT:PRINT:GOSUB9200
10040 S$="   FIN DE LA PARTIE":GOSUB9100
10042 END
10044 GOTO10042
10050 S$="ILS VOUS MENENT A LEUR CHEF":GOSUB9100
10052 S$="IL A DEJA RENCONTRE DES BLANCS":GOSUB9100
10054 S$="ET SE MONTRE TRES AMICAL":GOSUB9100
10056 GOSUB9200
10058 S$="VOS HOMMES SONT NOURRIS ET PEUVENT SE REPOSER":
GOSUB9100
10060 PRINT:GOSUB9200
10062 S$="LE LENDEMAIN LES ECHANGES COMMENCENT":GOSUB9100
10064 PRINT:GOSUB9200
10066 S$=K$:GOSUB9100
10068 GETI$:IFI$=""THEN10068
10069 RETURN
```

Variantes de basic

Spectrum :
Remplacez tout au long V1() par B(), V2() par D(), EQ(,) par Q(,) et AO() par E(), et procédez aux modifications suivantes :

```
60    DIM T$(3,9)
10001 CLS:GOSUB 9200
10022 INPUT I$:LET I$=I$( TO 1)
10068 LET I$=INKEY$:IF I$="" THEN GOTO 10068
```

BBC Micro :
Procédez aux modifications suivantes :

```
10001 CLS:GOSUB 9200
10068 I$=GET$
```

Ian McKinnell

GOTO END

Pour en finir ici avec le PASCAL, nous allons nous attarder sur divers aspects de ce langage que nous n'avons pas encore vus, en particulier sur la description des données.

Le créateur du PASCAL, Niklaus Wirth, avait intitulé un de ses livres *Structures de données + Algorithmes = Programmes*. Cela reflète bien l'importance de la description des données pour l'écriture fidèle d'algorithmes. Si nous utilisons, par exemple, un tableau de caractères (chars) pour représenter une chaîne de caractères, les fonctions et procédures nécessaires à leur traitement (pour trouver leur longueur, les concaténer, etc.) seront très différentes de celles qui auraient été nécessaires pour une liste liée.

Prenons par exemple la simple tâche de trouver la longueur d'une chaîne. Souvenez-vous que nous avons rempli les positions inoccupées de caractères ASCII nuls, chr(0), ou que nous avons conclu la liste dynamique par la valeur pointeur nulle (NIL). La version tableau est simple :

```
FUNCTION Longueur (S :chaîne):Cardinal;
  VAR
    N       :0..LongueurChaîne;
    trouvé  :booléen;
  BEGIN
    N :=0;
    trouvé :=faux;
    REPEAT
      N :=N+1;
      trouvé :=S[N]=chr (0)
    UNTIL trouvé OR (N=LongueurChaîne);
```

Programme TriArbre

Ce programme utilise quelques procédures du programme précédent Liste Cyclique. Elles sont signalées en lieu utile. Les données sont des noms, des montants de dettes, et toutes autres zones. Elles sont saisies au clavier et intégrées par ordre alphabétique dans un arbre binaire. Chaque nœud de l'arbre a deux pointeurs, pour les éléments de rangs inférieur et supérieur. L'arbre grandit en comparant les nouvelles données avec la zone nom de chaque nœud et en prenant la zone de liaison appropriée (BranchementInf ou BranchementSup). Lorsque nous trouvons un nœud vide (avec un branchement de valeur NIL), l'élément est mis en place. L'écriture des données au fichier se fait alors simplement par une procédure récursive. Vous pourriez utiliser ces exemples pour mettre en place un gestionnaire de bases de données très efficace — suite de programmes adaptés à vos propres besoins. Un mot néanmoins de mise en garde : lorsque vous introduisez un fichier trié dans un arbre binaire, chaque insertion se fait sur la même branche, et l'arbre sera une liste à un seul lien. Pour des applications plus simples, la liste cyclique est probablement plus universelle. Elle ne comporte aucune espèce de pointeur NIL, nous évitant ainsi un double test.

```
PROGRAM    TriArbre  ( entrées, sorties, FichierDonnées );

CONST
        NomFichier      = Donnéesarbres;
        LonsueurChaine  = 25;

TYPE
   Cardinal      = 0 .. EntierMax;
   TailleChaine  = 1 .. LonsueurChaine;
   Chaine        = PACKED ARRAY [ TailleChaine ]
                   OF char;
objet            = RECORD
                     Nom    :chaine;
                     {autres zones ..}
                     dette  : Cardinal
                   END;   {objet}

TypeFichier      = FILE OF objet;
arbre            =^branchement;
                   {** référence vers l'avant sur :}
branchement      = RECORD
                     élément          : objet;
                     BranchementAntérieur
                     BranchementPostérieur  : ombre
                   END;     {branchement}

VAR
    données       : objet;
    FichierDonnées  : TypeFichier;
    Tronc         : arbre
INCLUDE  'Utils.src'  {** Fichier source contenant :
                SauterBlancs, LireCarte, et Lireligne
                                                  **
               {11111111111111111111111111111111111111}
PROCEDURE  LireQuantité  (VAR  Quantité   : Cardinal );
        {** Validez une valeur Cardinale légale **}
  VAR
        OK       :  booléen;
  BEGIN
    REPEAT
      REPEAT
        écrire ( 'quantité ?' : 20 );
        IF EoLn ( entrées ) THEN
          ReadLn ( entrées );

        SauterBlancs ( entrées )
        {** recommencer lorsque fin de ligne **}
      UNTIL NOT EoLn ( entrée);
      LireCarte (entrées,quantité, OK);
    IF NOT OK THEN
      WriteLn ( '--- ERREUR ---' : 20,
                      ' recommencez la saisie' )
    UNTIL OK     ** le nombre doit être valide **
  END;    LireQuantité
          {11111111111111111111111111111111111111}
PROCEDURE      croissance  ( VAR  feuille   : arbre;
      ** greffer sur l'arbre **   données   : objet );
  BEGIN
     nouveau ( feuille );
WITH feuille ^DO
  BEGIN
    élément := données;
```

```
    IF trouvé
      THEN
        Longueur := N-1
      ELSE
        Longueur := LongueurChaîne
  END; {Longueur}
```

Ce programme prête à confusion. Pourquoi avons-nous besoin de la variable booléenne trouvé? Et pourquoi devons-nous assigner la valeur N-1 à Longueur? Ces détails, gênants, peuvent devenir très trompeurs avec des algorithmes plus importants s'il faut avoir recours à des stratagèmes artificiels de cette sorte.

L'utilisation de « drapeaux de bits » est sans doute le truc le plus vieux du programmeur, mais ils sont souvent introduits dans l'intérêt de l'ordinateur et non comme éléments naturels aux algorithmes. Il faut utiliser la variable locale N, et non Longueur qui est un identificateur de fonction et non pas une variable. Les identificateurs de fonctions présents dans la partie droite d'une instruction telle que :

```
Longueur := Longueur+1
```

essaieraient de lancer un appel récursif à la fonction Longueur elle-même.

Comparez cela à la fonction Longueur nécessaire aux chaînes utilisant une liste liée. Souvenez-vous que, avec la représentation dynamique, la définition TYPE de string (chaîne) est très différente, permettant la création d'une chaîne de longueur libre. De telles descriptions de données seront souvent définies récursivement.

De nombreux exemples simples utilisés pour illustrer la récursion pourraient être exprimés aussi bien (si ce n'est mieux) par des algorithmes itératifs. Mais voyons la fonction Longueur pour le type chaîne récursif :

```
FUNCTION Longueur (S :chaîne)     :Cardinal;
  BEGIN
    IF S=NIL
      THEN
        Longueur :=0
      ELSE
        Longueur :=succ (Longueur(S↑.suivant)
  END; {Longueur}
```

S est le début de la liste, et l'expression S↑.suivant sélectionne la zone pointeur du prochain enregistrement de la liste. Une alternative est de concevoir cette expression comme une liste commençant par l'enregistrement suivant (SaufPremier). A

```
    branchement antérieur := NIL;   ** Plus de données
    branchement postérieur   := NIL   après cette ramification
  END

END;   Croissance
       {11111111111111111111111111111111111111111111}

PROCEDURE          avancer      ( pousse  : arbre;
  ** trouver de la place pour insérer **  données : objet );
  VAR

      rameau     : arbre
      plusgrand  : booléen;
    BEGIN
      WHILE pousse <> NIL DO
        BEGIN      {** Trouver une branche vide **}
         rameau := pousse
         plusgrand := données          Nom>
                      rameau   élément   Nom;
         IF plusgrand
           THEN  {** avancer dans l'arbre **}
            pousse := pousse↑.   Branchementpostérieur
           ELSE  {** redescendre **}
            pousse := pousse↑.  Branchementantérieur
      END;

    Croissance ( pousse, données );
    IF plusgrand
      THEN      {** insérerapres **}
       rameau       BranchementPostérieur := pousse
                 ** insèreravant **
       rameau       BranchementAntérieur := pousse

  END;     Avances
           {11111111111111111111111111111111111111111111}

PROCEDURE        Print ( VAR    F      : TypeFichier;
  {** écrire les données à un fichier **}racine  : arbre );
  BEGIN
    IF racine <> NIL THEN
      WITH racine ↑ DO    {** récursion :}
        BEGIN   {** écrire dans l'ordre :}
     Print ( F, BranchementAntérieur );  {** Antérieur d'abord}
     write ( F, élément );   {la racine elle-même}
     Print ( F, BranchementPostérieur ) {Postérieurdernier **}
    END

  END;  {Print}
        {11111111111111111111111111111111111111111111}

PROCEDURE       reprendre  ( VAR racine     : arbre );
  {** Récupérer la mémoire allouée, pour d'autres usages **}
  BEGIN
```

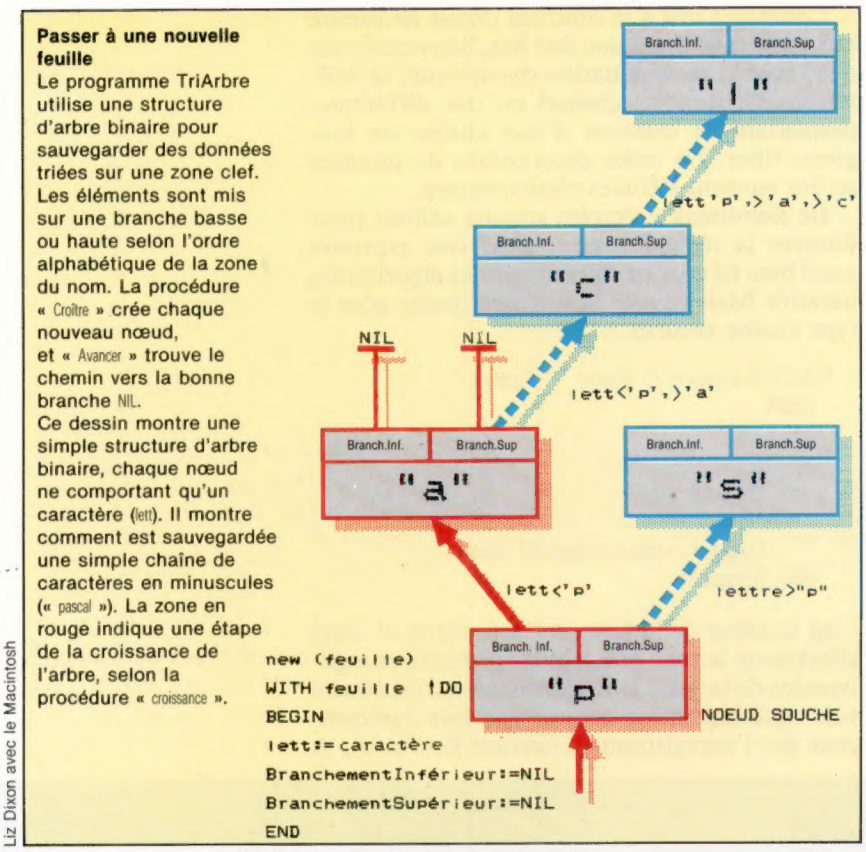
```
  <IF racine> NIL THEN  {** faire récursion **}
     BEGIN        ** récupérer les branches d'abord -
     Récupérer  ( racine    BranchementAntérieur );
     Récupérer  ( racine    BranchementPostérieur );
     Supprimer  ( racine )  - ensuite la racine **
    END

  END;   Récupérer
         {11111111111111111111111111111111111111111111}

BEGIN    TriArbre - Programme principal )

  assign ( FichierDonnées, NomFichier );
  page ( sorties );
  WriteLn ( '=== Tri de l'arbre ===' : 25 );
  WriteLn;
  WriteLn ( 'Saisir des données ( le nom d'abord)' );
  write ( 'Nom ? ');
  ReadLine ( données Nom );
  ReadAmount ( données . dette );
 croissance ( tronc, données );  * implanter le tronc de
              l'arbre *
  write ( Nom ? ' );
  ReadLine ( données . Nom );
 WHILE données . Nom  [ 1 ]<>chr ( 0 ) DO
   BEGIN
       LireQuantités ( données . date );
       Avancer ( tronc, données );
        WriteLn ( 'RETURN lorsque fait' : 40 );
        write ( 'Nom ? ' );
        ReadLine ( données . Nom )
    END;
    {** autres traitements, alors lire le fichier :}
  WriteLn ( 'Ecrire les données sur : ', NomFichier );
  rewrite ( FichierDonnées );
  Print ( FichierDonnées, Tronc );
  WriteLn;
  WriteLn ( 'Dette' : 8, 'Nom' : LongueurChaine );
  WriteLn;
  reset ( FichierDonnées );  {**  lire le fichier,}
          et afficher les données dans l'ordre :
 WHILE NOT EoF ( FichierDonnées ) DO
   BEGIN        ** lire chaque enregistrement **
        read ( FichierDonnées, données
          {** écrire les zones à la VISU :}
       WITH données DO
         WriteLn ( dette : 8, ' : ', Nom )
  END;

 Récupérer ( tronc );    {** vider la mémoire}
     pour d'autres traitements **
END.
```



Passer à une nouvelle feuille
Le programme TriArbre utilise une structure d'arbre binaire pour sauvegarder des données triées sur une zone clef. Les éléments sont mis sur une branche basse ou haute selon l'ordre alphabétique de la zone du nom. La procédure « Croître » crée chaque nouveau nœud, et « Avancer » trouve le chemin vers la bonne branche NIL.
Ce dessin montre une simple structure d'arbre binaire, chaque nœud ne comportant qu'un caractère (lett). Il montre comment est sauvegardée une simple chaîne de caractères en minuscules (« pascal »). La zone en rouge indique une étape de la croissance de l'arbre, selon la procédure « croissance ».

Liz Dixon avec le Macintosh

chaque rencontre de NIL, nous demandons une évaluation de la longueur de SaufPremier, et nous l'incrémentons par la fonction succ.

Beaucoup d'autres problèmes ne peuvent être résolus efficacement que par la récursion. L'exemple précédent est très simple et devrait vous permettre d'écrire facilement Longueur, sans utiliser la récursion. Cela prend plus de code, et il faut utiliser un compteur local (comme avec la version tableau) précisément pour éviter la récursion.

```
FUNCTION Longueur(S :chaîne)     :Cardinal;
  VAR
    N     :Cardinal;
  BEGIN
    N :=0;
    WHILE S <>NIL DO
      BEGIN
        N :=N+1;
        S :=S↑.suivant
      END;
    Longueur := N
  END; {Longueur}
```

Même avec cet algorithme itératif, la clarté et la concision ne sont qu'apparentes et dues à la simple nature récursive de la structure des données.

De nombreux compilateurs PASCAL mettent en œuvre des directives ou instructions pour le compilateur — et non des instructions ou déclarations. La seule directive demandée par le standard ISO est Forward. Lorsque deux procédures ou fonctions doivent s'entre-appeler, on dit qu'elles sont mutuellement récursives. Cela se produit rarement, et cela pose le problème de ne pouvoir utiliser un objet PASCAL quel qu'il soit avant qu'il n'ait été déclaré ou défini.

La solution est de ne déclarer que l'en-tête d'un des sous-programmes, remplaçant son bloc par la directive du compilateur FORWARD. Après définition complète de l'autre module, l'en-tête est donné en abrégé (sans la liste des paramètres) et le bloc est alors défini. On dispose quelquefois d'autres directives pour contrôler les options de compilation. Elles doivent cependant être évitées dans la mesure du possible, pour préserver la portabilité de vos programmes. Cela signifie que, outre la présence nécessaire d'un symbole spécifique (habituellement $) avant le premier caractère du commentaire, les options non portables ne sont pas susceptibles de recevoir un drapeau par d'autres compilateurs. Quelques versions non standards (notamment HiSoft) nécessitent une syntaxe légèrement différente pour les déclarations de pointeurs vers l'avant. Vous devez alors vous référer à votre manuel. Il y a peu de versions non standards, et il n'y en a aucune parmi les compilateurs PASCAL à la norme ISO.

Il existe une autre description des données en PASCAL que nous n'avons pas vue, la « variante ». Lorsque nous avons voulu stocker des éléments de type différent, nous avons utilisé un enregistrement approprié aux zones typées. Mais supposons que la description d'une partie de l'enregistrement, ou même de sa totalité, doive être modulable. Prenons l'exemple d'un fichier d'état civil : nous voulons stocker des informations personnelles sur des sujets donnés. L'attribution de ces informations dépendrait du fait que les sujets soient mariés ou non. Un enregistrement-variante a sa partie constante définie en premier. La partie variante est ensuite précisée par l'introduction d'un sélecteur de variante (de type simple), et par les mots réservés CASE et OR. Par exemple :

```
TYPE
  Sexe    = (masculin, féminin);
  variant = RECORD
    {toutes zones communes}
    CASE mariés : booléen OF
      faux :();
      vrai  :(DateMariage :chaîne;
              CASE sexe : sexe OF
                masculin : ();
                féminin : (NomDeJeuneFille :chaîne))
    END; {variante}
```

Remarquez que les listes de zones vides doivent toujours être mises entre crochets. L'espace d'un fichier est alors fixe (selon la plus grande variante), mais la mémoire peut être sauvegardée avec des pointeurs sur des variantes telles que new(p,vrai,mâle).

Les quelques points faibles du PASCAL ont été en grande partie supprimés par les spécialistes (et par Wirth dans MODULA-2). Ce langage est terriblement puissant bien que de petite taille. Il est très efficace, d'utilisation universelle et facile à apprendre. Les avantages d'avoir tous vos programmes PASCAL en code source portables de manière universelle sont inappréciables.

Compatible compétitif

L'un des premiers constructeurs de micros, la Tandy Corp., n'a pas véritablement réussi à s'imposer sur le marché. Le compatible Tandy 1000 marquera-t-il un tournant?

Chris Stevens

S'attaquer à un géant
Le Tandy 1000 est une machine compatible avec l'IBM PC sur laquelle son constructeur a misé pour reconquérir une bonne part du marché. Pour assurer la compatibilité, le Tandy 1000 est construit autour du processeur Intel 8088 et utilise les unités de disquette standards 5 $\frac{1}{4}$ pouce. Ici, le modèle, à deux unités de disquette, est muni de 128 K de mémoire additionnels.

Bien que la Tandy Corporation ait été l'un des pionniers de la micro-informatique avec le TRS-80, la société n'a pas réussi à remporter des succès comparables à ceux de Commodore dans le marché domestique, ou à ceux d'Apple dans le marché professionnel — tous deux étant ses concurrents de la première heure. Mais Tandy semble avoir acquis beaucoup d'expérience et a l'intention d'attaquer sérieusement le marché professionnel. D'une part, en s'associant avec ACT (le constructeur de l'Apricot), Tandy a étoffé la gamme des machines proposées dans sa chaîne européenne de centres informatiques, puisque ceux-ci vendent maintenant la gamme Apricot. D'autre part, la création de sa propre division de fabrication illustre le dynamisme de l'offensive de ce constructeur.

Il est de plus en plus certain que la clé du succès en micro-informatique consiste actuellement à proposer un matériel compatible avec l'IBM PC au meilleur prix possible. Aux États-Unis, c'est aussi vrai dans le marché domestique que dans le marché professionnel, puisque les consommateurs n'hésitent pas à acheter, pour la maison, des machines que les Européens jugent trop coûteuses pour être utilisées ailleurs qu'au bureau.

Le Tandy 1000 est proposé comme entièrement compatible; sa version de base inclut une unité de disquette et 128 K de mémoire pour un coût très compétitif. Tandy espère ainsi revitaliser sa division de produits informatiques.

La machine que nous examinons ici est une version 256 K à deux unités de disquettes. La machine ressemble un peu à l'IBM PC. L'unité consiste en un grand boîtier qui contient l'ordinateur lui-même, les interfaces et les lecteurs de disquette. Un moniteur est posé sur l'unité, et la mobilité du clavier permet de définir une position de travail très confortable.

D'apparence externe, le Tandy semble aussi robuste et aussi fiable que l'IBM PC. Comme celui-là, le Tandy possède un clavier détachable muni de touches moulées à disposition incurvée qui facilitent la frappe. De plus, afin d'assurer un meilleur confort de travail, deux pattes placées sous le clavier permettent de l'incliner à un angle de 15°.

Bien que le Tandy possède les touches nécessaires pour assurer *la* compatibilité, leur disposition est différente de celle du PC. Cette stratégie comporte ses inconvénients et ses avantages. Si le clavier du PC est généralement apprécié, il est — aussi — assez souvent critiqué en raison de l'étrange disposition de certaines touches vitales, comme les touches SHIFT et ALTERNATE. La taille relativement réduite des touches Control et RETURN a notamment été sévèrement critiquée.

Tandy a évidemment tiré des enseignements de ces critiques lors de la conception du clavier. La touche « / », qui compliquait l'utilisation de la touche SHIFT, a maintenant été retirée — sa fonction étant assurée par SHIFT-4 et SHIFT-7 sur le pavé numérique. De même, les touches ALTERNATE et CAPS LOCK ne sont plus placées de chaque côté de la barre d'espacement mais, respectivement, dans la zone du pavé

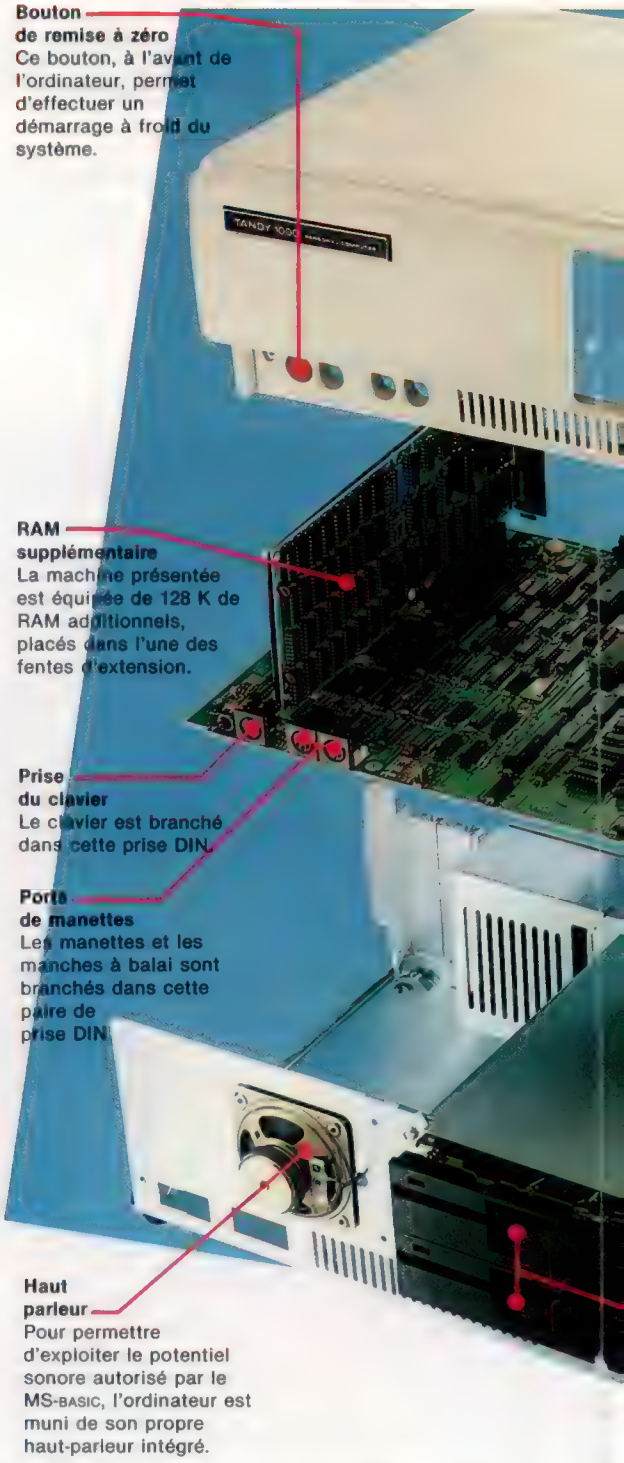
numérique et à l'extrême gauche sous les touches de commande. Ces changements donnent globalement une frappe plus confortable que sur le PC.

Le désavantage est évident : après avoir réussi à s'habituer à l'étrange clavier du PC, vous devrez réapprendre à utiliser cette autre disposition. Cependant, les nouveaux utilisateurs n'auront pas tous ces problèmes avec le clavier. Le mécanisme des touches lui-même est d'excellente qualité.

Tandy a aussi apporté d'autres changements au clavier. Les touches INSERT et DELETE ont été intégrées aux fonctions + et − et sont maintenant placées au-dessus du pavé numérique, et non en dessous. Parmi les touches additionnelles, mentionnons celles de déplacement de curseur individuelles pour l'édition, les touches HOLD, PRINT et BREAK, qui ont toutes été placées entre les touches texte et le pavé numérique. L'addition de ces touches impliquait un nouvel emplacement des touches de fonction, qui sont maintenant situées au-dessus des touches de texte. Pour inévitable qu'elle soit, cette nouvelle position est moins bonne que la précédente; mais à titre de compensation, Tandy a fourni deux touches de fonction additionnelles.

Pour ce qui est de l'ordinateur lui-même, Tandy a opté pour le large boîtier choisi par IBM. Les unités de disquettes 5 $\frac{1}{4}$ pouces sont situées à l'avant du côté gauche, bien que le modèle de base ne soit livré qu'avec une seule unité. Moins bruyantes que leurs équivalents IBM, les unités de disquette ont des vitesses d'accès légèrement différentes. Il est assez décevant que le dispositif de chargement ne soit pas à ressort. Cela signifie que les disquettes ne sont pas éjectées des unités lorsque vous les ouvrez et que vous devez les retirer manuellement.

L'arrière de la machine
En plus d'une interface imprimante Centronics, le Tandy 1000 possède deux types de prise de moniteur — RVB et vidéo composite. Afin d'utiliser pleinement le potentiel sonore de la machine, le Tandy est également muni d'un jack audio permettant d'amplifier les sons produits par l'ordinateur sur une chaîne hi-fi conventionnelle. (Cl. Chris Stevens.)



Bouton de remise à zéro
Ce bouton, à l'avant de l'ordinateur, permet d'effectuer un démarrage à froid du système.

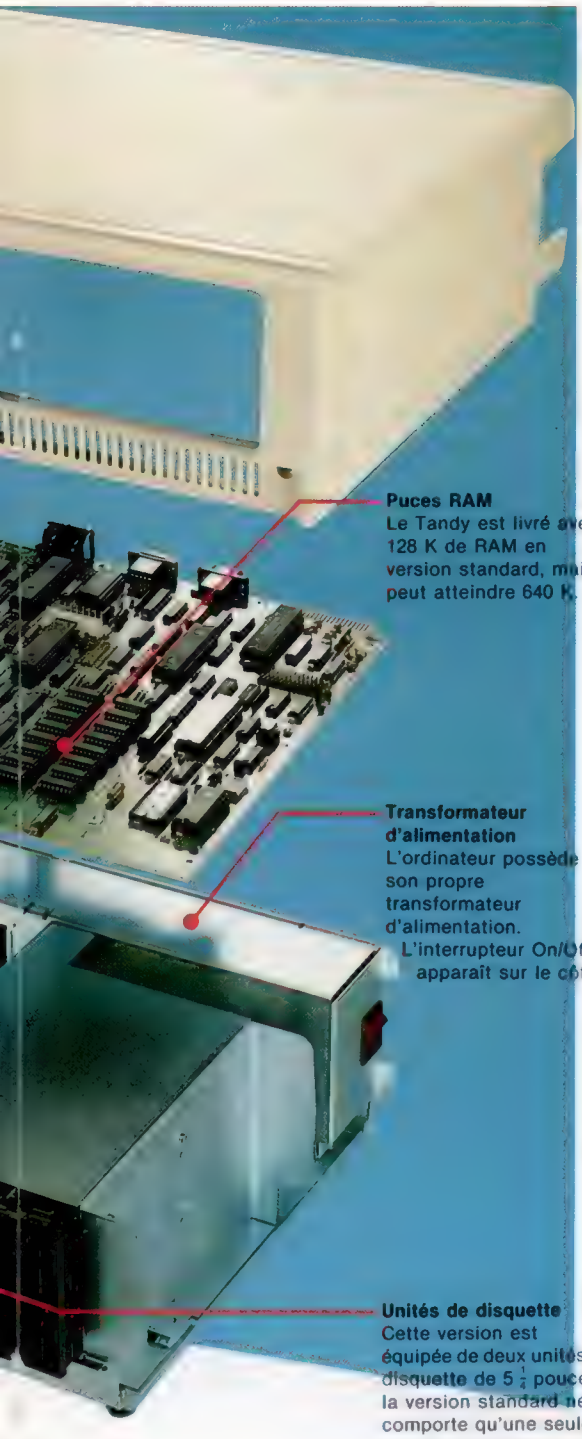
RAM supplémentaire
La machine présentée est équipée de 128 K de RAM additionnels, placés dans l'une des fentes d'extension.

Prise du clavier
Le clavier est branché dans cette prise DIN.

Ports de manettes
Les manettes et les manches à balai sont branchés dans cette paire de prise DIN.

Haut parleur
Pour permettre d'exploiter le potentiel sonore autorisé par le MS-BASIC, l'ordinateur est muni de son propre haut-parleur intégré.

Dans le coin inférieur gauche, sous une ouverture d'aération, une paire de connecteurs DIN à six broches est destinée au branchement de manches à balai ou à d'autres dispositifs de commande. A la gauche de ceux-ci, on aperçoit un gros bouton orange, le bouton de remise à zéro du système, qui permet d'effectuer un démarrage à froid. Bien qu'il soit trop apparent, il y a peu de risque qu'il soit pressé inopinément, puisqu'il se trouve en renfoncement dans le boî-



tier. En fait, il est très bien placé, puisque de nombreuses applications ne vous permettront de passer au système d'exploitation qu'en appuyant sur ce bouton. Il est donc pratique que le bouton de remise à zéro soit placé à un endroit facile à atteindre.

Les interfaces de périphériques sont situées à l'arrière de l'ordinateur. A l'extrême gauche, on aperçoit l'entrée de l'alimentation qui est située juste au-dessus de l'interface parallèle Centronics. A droite, un connecteur de type D à sept broches est destiné à un crayon optique et, sous ce connecteur, un port à sept prises permet de connecter un moniteur RVB. Plus à droite, sous un ventilateur de refroidissement, une paire de minijacks servent de prise vidéo composite pour des moniteurs vidéo composite monochromes ou couleurs et de prise audio permettant d'amplifier le son de l'ordinateur par l'intermédiaire d'un système hi-fi conventionnel. A l'extrême droite, trois prises d'extension permettent d'ajouter des interfaces de périphériques ou des modules de mémoire additionnels.

Comparaisons

Tandy insiste sur le fait que sa machine offre de nombreuses caractéristiques d'interface, qui sont absentes sur l'IBM PC. Même si cela est vrai, Tandy ne semble avoir prévu que le strict minimum pour une utilisation professionnelle. Par exemple, la version de base propose 128 K de RAM, ce qui n'est pas assez pour exécuter certains logiciels intégrés récents. Et bien que le Tandy 1000 soit une machine autonome, un port RS232 (qui est optionnel) est essentiel pour toute application de gestion sérieuse. En revanche, il représente un meilleur rapport qualité/prix que le PC qui, avec les mêmes caractéristiques, coûte beaucoup plus cher. Mais il est évidemment essentiel que la machine soit entièrement compatible avec le logiciel IBM.

Le problème pour réussir à être compatible avec l'IBM PC ne se situe pas au niveau de l'UC 8088 ou même du système d'exploitation MS-DOS, puisque tous deux peuvent être achetés auprès de leurs fabricants. La difficulté se situe au niveau du BIOS (Basic Input/Output System), dont IBM a les droits. De nombreux programmes utilisent des branchements directs à des routines BIOS, et si les routines correspondantes de la machine compatible ne sont pas situées aux mêmes endroits, le programme ne pourra fonctionner correctement. A cet égard, le BIOS du Tandy 1000, écrit par Phoenix Compatibility Corporation est exceptionnel. Non seulement le Lotus 1-2-3, très difficile, fonctionne sur la machine (bien que l'espace mémoire de 256 K paraisse serré), mais l'ordinateur exécute également Wordstar 2000 et dBASE II. Malgré cela, le Tandy 1000, comme l'IBM PC, est extrêmement lent dans les calculs mathématiques en BASIC. Un comptage de 0 à 1000, utilisé comme test de performances, mit 6 secondes — plus lent que sur plusieurs micros 8 bits.

Le Tandy 1000 représente sans aucun doute une solution intéressante pour l'utilisateur professionnel qui désire tirer parti de la quantité colossale de programmes de qualité qui ont été écrits pour l'IBM PC. Cependant, en raison des contraintes matérielles imposées par la compatibilité IBM — et par souci d'obtenir un coût de revient aussi bas que possible —, Tandy ne peut que proposer une machine qui est essentiellement démodée en terme d'informatique de gestion.

Cependant, à moins qu'IBM diminue le prix de ses machines (ce qui semble peu probable), Tandy Corporation vient là de produire un outil qui pourrait remporter un important succès auprès d'une large clientèle.

TANDY 1000

PRIX

★ ★ ★ ★ pour la version 128 K, avec une seule unité de disquette.

DIMENSIONS

420 × 335 × 150 mm.

UC

8088, fonctionnant à 4,77 MHz.

MÉMOIRE

128 K de RAM en version standard ; extension possible à 640 K.

ÉCRAN

80 × 25 caractères en mode texte, 640 × 200 points en mode haute résolution. La machine peut afficher jusqu'à 8 couleurs dans un choix de 16.

INTERFACES

Prises vidéo RVB et composite, port d'imprimante, jack audio et trois interfaces d'extension.

LANGAGES DISPONIBLES

BASIC, plus une sélection de langages offerts sous MS-DOS, dont COBOL, FORTRAN, etc.

CLAVIER

90 touches, dont un pavé numérique et 12 touches de fonction.

DOCUMENTATION

La documentation explique le fonctionnement de MS-DOS, de MS-BASIC et du logiciel intégré Deskmate fourni avec la machine.

FORCES

Entièrement compatible avec l'IBM PC, avec de nombreuses interfaces qui ne sont pas fournies sur le modèle de base du PC.

FAIBLESSES

Tandy n'a pas amélioré énormément la conception de la machine par rapport au PC. L'utilisation du processeur 8088 au lieu du 8086, plus perfectionné, signifie que le Tandy fonctionne plus lentement qu'il ne le devrait.

Exocet SUR EXL 100

Le micro EXL 100 d'Exelvision n'est pas oublié dans notre rubrique « Jeux ». Des programmes écrits par Pierre Monsaut ont été adaptés à cet ordinateur. En voici un.

Un porte-avions ennemi s'est aventuré dans les eaux territoriales et refuse d'obéir aux sommations. Aux commandes de votre Mirage 2000, vous devez absolument le détruire avant qu'il ne menace votre base. Tapez une touche quelconque pour tirer.

```
100 REM **********
110 REM * EXOCET *
120 REM **********
130 R=0
140 GOSUB 970
150 GOSUB 790
160 CALL COLOR("1MC")
170 LOCATE (AY,AX):PRINT A$;
180 IF BX>36 THEN LOCATE (BY,37):PRINT M
$;:BB=1:GOTO 210
190 CALL COLOR("1BC")
200 LOCATE (BY,BX):PRINT B$;
210 AX=AX-1
220 IF AX<1 THEN LOCATE (AY,1):PRINT M$;
:AX=38
230 BB=BB+.2
240 BX=INT(BB)
250 CALL KEY1(D3,D4)
260 IF D3<>255 AND EY=0 THEN EX=AX:EY=AY
+1:NX=NX-1
270 IF EY<>0 THEN 310
280 FOR I=1 TO 10
290 NEXT I
300 GOTO 160
310 EX=EX-1
320 EY=EY+1
330 LOCATE (EY-1,EX+1):PRINT N$;
340 IF EX<1 THEN EX=38
350 IF EY=23 THEN 400
360 IF EY=22 AND ABS(EX-2-BX)<2 THEN GOS
UB 620
370 CALL COLOR("1RC")
380 LOCATE (EY,EX):PRINT E$;
390 GOTO 160
400 IF EX=40 THEN EX=1
410 LOCATE (EY-1,EX+1):PRINT N$;
420 EY=0
430 EX=0
440 IF NX=0 THEN 460
450 GOTO 160
460 CLS
470 IF S>R THEN R=S
480 CALL KEY1(D3,D4)
490 IF D3<>255 THEN 480
500 CALL COLOR("1BC")
510 LOCATE (10,11)
520 PRINT "SCORE :";S;
530 LOCATE (13,11)
540 PRINT "RECORD :";R;
550 LOCATE (16,11)
560 PRINT "UNE AUTRE ?";
570 CALL KEY1(D3,D4)
580 IF D3=255 THEN 570
590 IF D3<>78 THEN 150
600 CLS
610 END
620 LOCATE (EY-1,EX+1)
630 PRINT N$;
640 S=S+10
650 LOCATE (EY,EX)
660 CALL COLOR("1bC")
670 PRINT F$;
680 FOR I=1 TO 30
690 X=INTRND(4)-2
700 Y=INTRND(6)-1
710 LOCATE (EY-Y,EX-X)
720 PRINT F$;
730 NEXT I
740 FOR I=1 TO 200
750 NEXT I
760 NX=NX+1
770 CLS
780 GOTO 160
790 CLS
800 B$=CHR$(32)&CHR$(100)&CHR$(101)&CHR$
(102)
810 AX=38
820 S=0
830 BB=1
840 BX=1
850 A$=CHR$(103)&CHR$(104)&CHR$(32)
860 N$=CHR$(32)
870 M$=N$&N$&N$
880 E$=CHR$(105)
890 F$=CHR$(106)
900 EX=0
910 EY=0
920 XC=2
930 NX=20
940 BY=22
950 AY=8
960 RETURN
970 CLS ("BCb")
980 CALL CHAR(100,"000000000007FFFF7F00"
)
990 CALL CHAR(101,"0020203BFCFFFFFFFF00"
)
1000 CALL CHAR(102,"0000000000E0FFFCFB00
")
1010 CALL CHAR(103,"0000000000003F7FFF00
")
1020 CALL CHAR(104,"000000000103FFFFFF00
")
1030 CALL CHAR(105,"00000000007DFF7D0000
")
1040 CALL CHAR(106,"000B21B00A002B001000
")
1050 RETURN
```

Connaître votre micro

Nous avons déjà présenté des livres permettant de mieux exploiter les possibilités des micros Thomson. Nous donnons ici une liste de livres pour le VG 5000 de Philips et l'Alice de Matra.

Vous avez acquis un ordinateur avant d'aborder cette encyclopédie, ou bien vous avez fait cet achat au cours de sa lecture. Peut-être avez-vous déploré que l'on parle peu des particularités de votre machine, et vous êtes-vous senti un peu seul à parler votre « dialecte » BASIC.

Mais rassurez-vous. Pour tous les micro-ordinateurs présents sur le marché, il existe une bibliographie plus ou moins abondante. Aussi avons-nous sélectionné pour vous quelques titres d'ouvrages consacrés aux matériels les plus répandus en France. Ils vous aideront à maîtriser parfaitement votre ordinateur et, dès lors, vous serez à même d'adapter la plupart des programmes présentés dans *ABC Informatique* à votre machine.

Philips

VG5000 pour tous Initiation + programmes

Le VG 5000 de Philips est un petit micro, surtout destiné aux jeunes utilisateurs désireux de s'initier à la programmation en BASIC et de s'y perfectionner. L'ouvrage est conçu de façon à faciliter cet apprentissage à travers des exemples très simples et progressifs. La plupart des instructions disponibles sont traitées soit par un organigramme, soit par un texte.
Elles sont regroupées dans un index en fin d'ouvrage.

Par J.-M. Jégo.
145 pages, format 17 × 23 cm.
P.S.I.

102 programmes pour VG 5000

La plupart de ces jeux destinés à l'ordinateur individuel VG 5000 sont utilisables sans grande modification sur d'autres ordinateurs utilisant aussi un BASIC Microsoft. Les programmes proposés sont très courts et classés par ordre de difficulté croissante, suivant cinq niveaux. Chaque jeu est présenté par une courte description, suivie de l'étude détaillée des lignes importantes du programme et, enfin, quelques suggestions pour des améliorations ou des modifications permettant de rendre le jeu plus attrayant.

Par J. Deconchat.
240 pages, format 17 × 25 cm.
P.S.I.

Matra

20 programmes astucieux pour Alice

Ce recueil illustré de photos et dessins vous propose une grande variété de programmes de jeux à exécuter sur votre micro-ordinateur Alice : jeux éducatifs, divertissements populaires connus ou plus originaux, qui tous feront appel autant à vos réflexes qu'à votre sens stratégique, et aideront le programmeur débutant à concevoir ultérieurement de nouveaux programmes de son cru.

Par I. Creasey et A. Stemmer.
95 pages, format 14 × 22 cm.
Hachette Informatique.

Philips

Matra

Jeux sur VG 5000 Philips

Des techniques générales de programmation des jeux, faisant surtout appel au graphisme, précèdent la présentation des jeux proprement dits, et sont classés par catégories : action à un ou deux joueurs, réflexion à un ou deux joueurs, aventure. Chaque jeu est présenté par un organigramme et une étude du programme, suivis du listage. Ce livre de B. Amsler et O. Villemaud a été réalisé en étroite collaboration avec Philips.

215 pages, format 15 × 21 cm.
Edimicro.

Pratique du micro-ordinateur Alice

Entrez, vous aussi, dans ce monde merveilleux que vous ouvre l'informatique. Grâce à ce livre, passez de l'autre côté du miroir et apprenez tout ce que vous devez savoir pour mettre votre micro-ordinateur à votre service. Ce livre vous montre comment, dans un langage simple et accessible à tous; il est abondamment illustré d'exemples pratiques, totalement développés, avec leurs résultats, et clairement commentés. Ce sera votre guide le plus sûr.

Par H. Lilen.
160 pages; format 21 × 29,7 cm.
Editions Radio.

Matra

Matra

Dessiner, peindre... et jouer avec Alice

Apprendre à programmer en dessinant, en coloriant et en jouant, tel est l'objectif de ce livre qui réussit ainsi à joindre l'utile à l'agréable. Vous pourrez faire du dessin à main levée et créer des effets graphiques, apprendre la géographie, programmer de nombreux jeux faisant appel à votre intelligence et à votre habileté. Vous aurez aussi appris à programmer en BASIC Microsoft, un standard du marché.

Par L. Gros.
152 pages, 17 × 22 cm.
Eyrolles.

Philips

Cadrage

Nous explorons quelques-unes des routines en virgule flottante d'interpréteur BASIC pour le C64, en développant un programme graphique sur écran haute résolution.

Les routines en virgule flottante de l'interpréteur BASIC ne sont pas très bien documentées. Par exemple, il n'y a pas de table de saut des adresses d'appel, comme dans le cas du noyau. Il sera donc long de trouver les appels dont vous aurez besoin. Aussi, pour plus de précision, nous tenterons de faire une routine graphique en langage machine qui nous permettra de faire tourner une figure « fil de fer » en trois dimensions sur l'écran haute résolution.

Les techniques mathématiques dont il est question ici pourraient, bien sûr, servir de base à d'autres routines arithmétiques rapides, telles que la multiplication matricielle. Il est bon de se souvenir qu'en faisant des calculs arithmétiques en temps réel, afin de créer des images d'écran successives, il n'est pas nécessaire d'avoir la meilleure approche en programmation. Il est souvent préférable de calculer d'abord les données à partir desquelles seront affichées les différentes figures, avant de commencer la séquence d'animation. Toutefois, la technique de temps réel sert plus efficacement notre but.

Les variables BASIC sont stockées en mémoire au-dessus du programme. Ces variables sont contenues dans une Table de Variables, dont le début est indiqué par le contenu des emplacements 45 et 46 (décimal). Comme tous les pointeurs dans le C64, ceux-ci sont stockés en format lo-hi. L'adresse de départ de la Table des Variables est donc indiquée par la formule :

```
PEEK(45) + 256*PEEK(46)
```

Les divers pointeurs associés aux variables et leurs contenus normaux sont indiqués dans le tableau suivant :

Pointeur	Fonction	Contenu normal
43/44	Début de BASIC	2049
45/46	Début de variables	Dépend de la longueur du programme
47/48	Début de tableaux	Dépend du nombre de variables
49/50	Fin de tableaux + 1	Dépend de nombre/taille de tableaux
51/52	Bas de chaînes en cours	Dépend de nombre/taille des chaînes
55/56	Haut de mémoire	40960

Notez que les chaînes dynamiques sont construites à partir du haut de la mémoire au fur et à mesure qu'elles sont définies. Toutefois, les tableaux sont stockés au-dessus des autres variables dans la Table des Variables et, lorsqu'on rencontre une nouvelle variable au cours de l'exécution du programme, le système d'exploitation décale toute la zone de stockage de tableau du nombre d'octets requis pour la stocker.

Le stockage de chaînes est nécessairement plus compliqué que celui d'autres variables. Les variables entières (pas dans un tableau) et les variables en virgule flottante requièrent toutes sept octets, mais une chaîne peut en nécessiter jusqu'à 255. Pour venir à bout de cette complication, le système d'exploitation ne stocke que la longueur de la chaîne et un pointeur à son adresse de début dans la Table de Variables. Si une chaîne est définie dans un programme BASIC (A$="ABCD", par exemple), alors ce pointeur indiquera le premier octet de la chaîne dans la zone programme BASIC. Une telle chaîne est appelée « statique ». Si le programme modifie la valeur d'une chaîne, on dit qu'elle est « dynamique ». Les valeurs de chaînes dynamiques sont construites en descendant à partir du haut de la mémoire, et le pointeur dans la Table de Variables est modifié en conséquence. Ainsi, chaque entrée est contenue dans les mêmes sept octets.

L'adresse d'une variable dans la Table de Variables peut être facilement trouvée à partir de BASIC. Le secret réside dans le fait que, juste après qu'une variable a été appelée par l'interpréteur BASIC, son adresse sera indiquée par le pointeur de page zéro dans les adresses décimales 71 et 72, et cela peut être inspecté pour découvrir les adresses. Toutefois, nous devons rapidement récupérer cette information puisque l'emplacement 71 est aussi utilisé par le système d'exploitation lorsqu'il évalue certaines expressions numériques.

Les programmes suivants illustrent le format des variables ordinaires telles qu'elles sont stockées en mémoire. La première routine récupère une chaîne de la mémoire. Ici, nous utilisons la technique que nous venons de décrire pour inspecter les contenus des emplacements 71 et 72 et stocker immédiatement ces valeurs dans deux emplacements de réserve dans le tampon cassette. Ceux-ci serviront à calculer ultérieurement l'adresse de la chaîne, afin de la récupérer.

```
1000 REM**TROUVER UNE CHAINE EN MEMOIRE**
1010 X$="ABCDEF"
1020 REM**FAIRE DE X$ UNE VARIABLE EN COURS**
1030 X$=X$+" "
1040 REM**SAUVEGARDER POINTEUR VERS TABLE VAR**
1050 POKE 828,PEEK(71):POKE 829,PEEK(72)
1060 REM**ADR DANS TABLE VAR**
1070 ADR=PEEK(828)+256*PEEK(829)
1080 REM**REGARDER ENTREE DANS TABLE VAR**
1090 LS=PEEK(ADR):REM LONGUEUR DE CHAINE
1100 SA=PEEK(ADR+1)+256*PEEK(ADR+2)
1120 REM**LECTURE DE CHAINE**
1130 FOR I=SA TO SA+LS
```

```
1140 VAR$=VAR$+CHR$(PEEK(I))
1150 NEXT
1160 PRINT VAR$
```

Vous pensez peut-être qu'en utilisant des variables entières (marquées par un signe pour cent, tel que X%) vous économiserez de la mémoire et accélérerez le calcul. Avec le C64, toutefois, ce n'est pas le cas. Lorsque le BASIC Commodore doit effectuer des calculs sur des entiers, il les convertit en virgules flottantes et appelle les routines virgule flottante ! Ainsi, bien que les variables entières puissent être stockées sur deux octets seulement, à moins d'être stockées dans un tableau, elles se verront attribuer chacune sept octets d'espace mémoire. Ces octets supplémentaires seront simplement ignorés au cours du traitement. La routine suivante vous permet de localiser un entier en mémoire.

```
1000 REM**TROUVER UN ENTIER EN MEMOIRE**
1010 X%=3456
1020 REM**FAIRE DE X% UNE VARIABLE EN COURS**
1030 X%=X%
1040 REM**SAUVEGARDER POINTEUR VERS TABLE VAR**
1050 POKE 828,PEEK(71):POKE 829,PEEK(72)
1060 REM**ADR DANS TABLE VAR**
1070 ADR=PEEK(828)+256*PEEK(829)
1080 REM**REGARDER ENTREE DANS TABLE VAR**
1090 LO=PEEK(ADR+1):HI=PEEK(ADR)
1100 REM**CALCULER RESULTAT**
1110 SIGNBIT=(HI AND 128)/128
1120 VAR=LO+256*(HI AND 127)-32768*SIGNBIT
1130 PRINT VAR
```

La même technique d'emplacement d'adresse peut aussi être utilisée pour une variable en virgule flottante. Il existe cependant une méthode bien plus économique, basée sur le fait que lorsque DEF FN sert à définir une fonction de la variable X, alors X est utilisé (mais sa valeur reste inchangée) chaque fois que FN est appelé.

Le programme suivant utilise DEF FN pour calculer l'adresse de la variable BASIC en cours, contenue dans les emplacements 71 et 72. Comme X est utilisé comme une variable fonction, cela assure que l'adresse générée est celle du premier octet de X contenu dans la Table de Variables BASIC. FN est alors appelé, attribuant cette adresse à la variable ADD. Notez que le passage d'un zéro (ou d'une autre valeur) via la commande FN ne change pas la valeur de X contenue dans la Table de Variables, ni l'adresse calculée.

```
1000  REM**TROUVER FPVAR EN MEMOIRE**
1010 DEF FNADR(X)=PEEK(71)+256*PEEK(72)
1020 ADD=FNADR(0):REM RETOURNE TOUJOURS A L'
ADRESSE DE X
1030 X=3.14159
1040 REM**CONVERTIR    BASE 2 EN DECIMAL**
1050 POWERTO=2^(PEEK(ADD)-129)
1060 SIGN=(-1)^((PEEK(ADD+1)AND 128)/128)
1070 REM*PARTIE FRACTION LARGE DE 31 BITS*
1080 D1=PEEK(ADD+1)AND 127:REM 7 BITS
1090 D2=PEEK(ADD+2):REM 8 BITS
1100 D3=PEEK(ADD+3):D4=PEEK(ADD+4)
1110 REM** GULP! **
1120 FRACT=2^(-7)*D1+2^(-15)*D2+2^(-23)
*D3+2^(-31)*D4
1130 MANT=1+FRACT
1140 VAR=SIGN*POWERTWO*MANT
1150 PRINTVAR
```

Lorsqu'une instruction de DIMension est exécutée, de la mémoire est réservée pour le tableau. Cela consiste en un en-tête de tableau, plus le nombre d'octets nécessaires pour le stockage de l'élément. Le format des éléments stockés dans un tableau est différent pour chacun des types de variables que nous avons vus ici. Considérons maintenant l'arithmétique en virgule flottante. Lorsque l'interpréteur BASIC effectue des calculs en virgule flottante, il stocke tous les résultats intermédiaires dans deux « accumulateurs de virgules flottantes ». Ceux-ci sont généralement appelés respectivement FAC et ARG, et le format utilisé est le même que celui qui sert à stocker des variables en mémoire. FAC se trouve aux adresses $61 à $65 (97 à 101 décimal), et ARG utilise $69 à $6D (105 à 109 décimal). Pour simplifier, nous continuerons à utiliser les routines interpréteur qui ne font que brasser des nombres entre FAC et la mémoire. Les routines interpréteur avec lesquelles nous allons nous familiariser dans cette partie sont les suivantes :

• MOVFM (adresse d'appel $BBA2) :
La fonction de cette routine est de charger le contenu de FAC à partir d'une variable virgule flottante en mémoire. Elle est symboliquement représentée par F ← M. Pour l'appeler, charger l'accumulateur avec l'octet lo de l'adresse de début de variable, et le registre Y avec l'octet hi.

• MOVMF (adresse d'appel $BBD4) :
Cette routine met le contenu de FAC dans sept octets en mémoire, soit M ← F. Pour l'appeler, charger le registre X avec l'octet lo et le registre Y avec l'octet hi de la destination de l'octet de début de variable en mémoire.

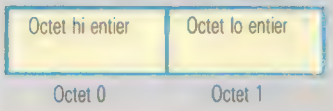
• FMULT (adresse d'appel $BA28) :
C'est la routine de multiplication qui multiplie le contenu de FAC par une seconde variable en mémoire, et stocke le résultat en FAC. Nous plaçons la première variable dans FAC à l'aide de MOVFM et pointons la seconde variable en chargeant l'accumulateur avec l'octet lo et le registre Y avec l'octet hi de l'octet de début, avant d'appeler cette routine. Finalement, nous pouvons, si nécessaire, remettre le résultat en mémoire à l'aide de MOVMF.

• FADD (adresse d'appel $B867).
Cette routine d'addition effectue FAC=MEM−FAC. Pour l'appeler, nous devons charger l'accumulateur avec l'octet lo de MEM et le registre Y avec l'octet hi de MEM.

• FSUB (adresse d'appel $B850) :
Cette routine de soustraction effectue FAC=MEM−FAC. Pour l'appeler, nous chargeons l'accumulateur avec l'octet lo de MEM et plaçons

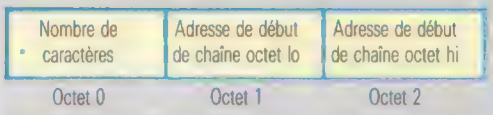
Variables stockées dans un tableau

Variables entières



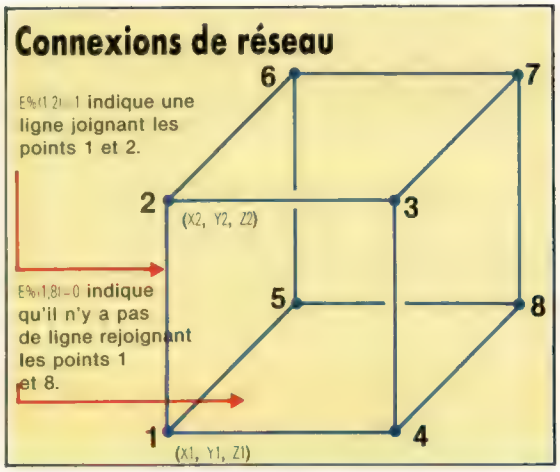
Variables virgule flottante

Variables chaînes



Types de variables
Il existe trois types de variables de tableaux sur le C64, chacun d'un format différent lorsqu'ils sont contenus en octets dans la mémoire. Les variables entières sont contenues dans deux octets comme des nombres compléments à deux. Les nombres en virgule flottante requièrent cinq octets pour contenir la mantisse, un bit de signe et l'exposant. Les variables chaînes sont contenues dans une autre zone de mémoire, utilisant un octet pour chaque caractère dans la chaîne. Toutefois, cette donnée n'est pas contenue dans la table de tableaux ; par contre, trois octets sont utilisés pour donner le nombre de caractères dans la chaîne et l'adresse à 16 bits qui pointe sur le début de la zone donnée caractères de chaîne.

Fils de fer partout
Le programme ROTSUB fait tourner une figure en fil de fer sur l'écran haute résolution du C64. Cette figure est définie à l'aide de quatre tableaux, dont trois servent à contenir les coordonnées *x*, *y* et *z* de chaque nœud.
Un quatrième tableau — E% — est utilisé pour définir si deux nœuds particuliers sont ou non reliés par une ligne. Dans cet exemple, les quatre tableaux sont initialisés pour produire une figure cubique.



l'octet hi de MEM dans le registre Y. Justement, ce sont les routines d'interpréteur que nous allons construire dans la première phase de notre programme graphique. Considérons à présent les composantes de ce programme. L'idée sous-jacente de ce projet est qu'une figure « fil de fer » peut être spécifiée par un certain nombre de points (ou nœuds) et une matrice de connexion des arêtes. Les nœuds ont pour coordonnées X(I), Y(I), Z(I) — où I va de 1 à NP (le nombre de points). La matrice des connexions d'arêtes est E%(I,J) — où I et J sont compris entre 1 et NP. Nous définissons E%(I,J) comme égal à 1 si le point I est connecté au point J, et zéro sinon. Ce n'est évidemment pas très économique en mémoire, parce que nous utilisons deux octets là où un seul suffirait. Mais il est plus facile pour l'utilisateur de spécifier quels points doivent être connectés. De plus, dans les applications pratiques, NP ne sera pas très grand. Pour communiquer avec la routine de rotation en langage machine à partir de BASIC, nous POKons les adresses de base des tableaux commençant par l'élément 1. Ainsi, les adresses de X(1), Y(1), Z(1) et E%(1,1) n'auront pas besoin d'être trouvées. En outre, nous suppléerons au langage machine avec le nombre de points, NP, et les COS et SIN de l'angle de rotation requis autour de l'axe *z*.

Nous avons planifié le projet en trois phases. D'abord, nous définissons les algorithmes et écrivons un programme test en BASIC. C'est fait par le « programme de Rotation de Cube » donné ici, qui fait tourner un cube autour de l'axe *z* et projette le résultat sur le plan *x*, *y*. Notre objectif est de convertir le listage BASIC en langage machine. Mais nous commençons par la tâche la moins ambitieuse consistant à traduire le sous-programme à la ligne 1800 de la version en langage machine. C'est fait par le programme I-ROTSUB, et un programme test pour cette routine est donné.

Programme I-ROTSUB 64

Le listage d'assemblage qui suit doit être assemblé et chargé, et le code résultant sauvegardé comme I-ROT.HEX.

```
;++++++++++++++++++++++++++++
;++     I-ROTSUB 64        ++
;++                        ++
;++++++++++++++++++++++++++++
;++USES ARRAYS DEFINED     ++
;++FROM BASIC ADDRESSES    ++
;++MUST BE POKED FIRST     ++
;++++++++++++++++++++++++++++
;
;PLTSUB = $C183
;XLO    = $C103
;XHI    = $C104
;YLO    = $C105
;
*       = $C544
;
;++++ ROTSUB  VARIABLES ++++
;
; VARIABLES CALLED FROM BASIC
XBASLO  *=*+1 ; POKE50500,X(0)LO
XBASHI  *=*+1 ; POKE50501,X(0)HI
YBASLO  *=*+1 ; POKE50502,Y(0)LO
YBASHI  *=*+1 ; POKE50503,Y(0)HI
NP      *=*+1 ; POKE50504,NP
; VARIABLES USED BY M/C
XILO    *=*+1
XIHI    *=*+1
YILO    *=*+1
YIHI    *=*+1
CSLO    *=*+1
CSHI    *=*+1
SNLO    *=*+1
SNHI    *=*+1
MEM1    *=*+5 ;FLOATING POINT VAR
MEM2    *=*+5 ;FLOATING POINT VAR
FAC     =$0061
ARG     =$0069
;
;INTERPRETER ARITHMETIC CALLS
;
FMULT   =$BA28 ; FAC=FAC*ARG
FADDT   =$B86A ; FAC=FAC+ARG
FSUB    =$B850 ; FAC=MEM-FAC
FADD    =$B867 ; FAC=FAC+MEM
MOVFM   =$BBA2 ; FAC=MEMORY
MOVMF   =$BBD4 ; MEMORY=FAC
;
;++++ SAVE REGISTERS++
;
        PHA
        TXA
        PHA
        TYA
        PHA
;
;++++ INITIALISE VARIABLES ++++
;
        LDA XBASLO
        STA XILO
        LDA XBASHI
        STA XIHI
        LDA YBASLO
        STA YILO
        LDA YBASHI
        STA YIHI
;
;++++PERFORM  MEM1=X(I)*CS-Y(I)*SN
;
START
        LDA XILO
        LDY XIHI
        JSR MOVFM   ; FAC = X(I)
        LDA CSLO
        LDY CSHI
        JSR FMULT   ; FAC = X(I)*CS
        LDX #<MEM1
        LDY #>MEM1
        JSR MOVMF   ; MEM1=X(I)*CS
        LDA YILO
        LDY YIHI
        JSR MOVFM   ; FAC = Y(I)
        LDA SNLO
        LDY SNHI
        JSR FMULT   ; FAC = Y(I)*SN
        LDA #<MEM1
        LDY #>MEM1
        JSR FSUB    ; FAC = MEM1-FAC
        LDX #<MEM1
        LDY #>MEM1
        JSR MOVMF   ; MEM1= FAC
;
;++++PERFORM MEM2=Y(I)*CN+X(I)*SN
;
        LDA YILO
        LDY YIHI
        JSR MOVFM   ; FAC = Y(I)
        LDA CSLO
        LDY CSHI
        JSR FMULT   ; FAC = Y(I)*CS
        LDX #<MEM2
        LDY #>MEM2
        JSR MOVMF   ; MEM2= Y(I)*CS
        LDA XILO
        LDY XIHI
        JSR MOVFM   ; FAC = X(I)
        LDA SNLO
        LDY SNHI
        JSR FMULT   ; FAC = X(I)*SN
        LDA #<MEM2
        LDY #>MEM2
        JSR FADD    ; FAC = MEM2+FAC
        LDX #<MEM2
        LDY #>MEM2
        JSR MOVMF   ; MEM2= FAC
;
;++++PERFORM X(I) =MEM1:Y(I)=MEM2
;
        LDA #<MEM1
        LDY #>MEM1
        JSR MOVFM   ; FAC =MEM1
        LDX XILO
        LDY XIHI
        JSR MOVMF   ; X(I)=FAC
        LDA #<MEM2
        LDY #>MEM2
        JSR MOVFM   ; FAC =MEM2
        LDX YILO
        LDY YIHI
        JSR MOVMF   ; Y(I)=FAC
;
;++++TEST END LOOP
;
        DEC NP
        BEQ EXIT
;
;++++INCREMENT ARRAY POINTERS
;
        LDA #$05
        CLC
        ADC XILO
        STA XILO
        BCC XNOHI
        INC XIHI
XNOHI
        LDA #$05
        CLC
        ADC YILO
        STA YILO
        BCC YNOHI
        INC YIHI
YNOHI
        JMP START
;
;++++ PULL REGISTERS OFF STACK ++++
;
EXIT
        PLA
        TAY
        PLA
        TAX
        PLA
;
        RTS
.END
```

Programme basic de rotation du cube

```
1000 REM** BASIC ROTATING CUBE**
1010 IFA=0THENA=1:LOAD"PLOTSUB.HEX",8,1
1020 IFA=1THENA=2:LOAD"LINESUB.HEX",8,1
1030 REM**DIMENSION ARRAYS**
1040 NP=8:REM NUMBER OF POINTS
1050 DIM X(NP),Y(NP),Z(NP)
1060 DIM ED(NP,NP):REM EDGE CONNECTIONS
1070 REM**INITIALISE ARRAYS**
1080 REM--CUBE COORDINATE DATA
1090 DATA  75, 75, 75:REM----------/1
1100 DATA -75, 75, 75:REM TOP FOUR /2
1110 DATA -75,-75, 75:REM POINTS   /3
1120 DATA  75,-75, 75:REM----------/4
1130 DATA  75, 75,-75:REM----------/5
1140 DATA -75, 75,-75:REM BOT FOUR /6
1150 DATA -75,-75,-75:REM POINTS   /7
1160 DATA  75,-75,-75:REM----------/8
1170 REM**ROTATE CUBE ABOUT X-AXIS π/4
1180 FORI=1TONP
1190 READX(I),Y(I),Z(I)
1200 Y(I)=Y(I)*COS(π/4)-Z(I)*SIN(π/4)
1210 Z(I)=Z(I)*COS(π/4)+Y(I)*SIN(π/4)
1220 NEXT
1230 REM**ROTATE CUBE ABOUT Z-AXIS π/4
1240 FORI=1TONP
1250 X(I)=X(I)*COS(π/4)-Y(I)*SIN(π/4)
1260 Y(I)=Y(I)*COS(π/4)+X(I)*SIN(π/4)
1270 NEXT
1280 REM--EDGE CONNECTION DATA--
1290 E(1,2)=1:REM 1 CONNECTED TO 2
1300 E(2,3)=1:E(3,4)=1:E(4,1)=1
1310 E(5,6)=1:REM BOT SQUARE
1320 E(6,7)=1:E(7,8)=1:E(8,5)=5
1330 E(5,1)=1:REM TOP TO BOT EDGES
1340 E(6,2)=1:E(7,3)=1:E(8,4)=1
1350 REM**SYMMETRISE E(I,J)**
1360 FORI=1TONP:FORJ=1TONP
1370 IFE(I,J)=1THENE(J,I)=1
1380 NEXT:NEXT
1390 REM#########################
1400 REM##PLOT ROTATING CUBE##
1410 SA=2*π/45
1420 FOR A=π/4 TO π/4+2*π STEP SA
1430 GOSUB1800:REM ROTATE THRU SA
1440 GOSUB1590:REM INIT/CLEAR SCREEN
1450 REM--PLOT CUBE--
1460 FORI=1TONP
1470 FORJ=1TOI
1480 IF E(I,J)=0THEN1510:REM NOT JOINED
1490 GOSUB1630:REM COMPUTE PROJECTION
1500 GOSUB1670:REM JOIN POINTS
1510 NEXT:NEXT
1520 REM-------------
1530 NEXT A:REM NEXT ANGLE
1540 REM#########################
1550 REM**WAIT**
1560 GETA$:IFA$=""THEN1560
1570 GOSUB1760:REM RESET SCREEN
1580 END
1590 REM**SETUP HIRES**
1600 POKE49408,1:POKE49409,1
1610 POKE49410,1:SYS49422
1620 RETURN
1630 REM**COMPUTE PROJECTION ON HIRES**
1640 X1%=X(I)+159:Y1%=199-(Z(I)+100)
1650 X2%=X(J)+159:Y2%=199-(Z(J)+100)
1660 RETURN
1670 REM**LINESUB**
1680 IF(X1%=X2%)AND(Y1%=Y2%)THENRETURN
1690 MHI=INT(X1%/256):MLO=X1%-256*MHI
1700 NHI=INT(X2%/256):NLO=X2%-256*NHI
1710 POKE49920,MLO:POKE49921,MHI
1720 POKE49922,NLO:POKE49923,NHI
1730 POKE49924,Y1%:POKE49925,Y2%
1740 SYS49934:REM LINESUB
1750 RETURN
1760 REM**RESET SCREEN**
1770 POKE49408,0:SYS49422
1780 PRINTCHR$(147)
1790 RETURN
1800 REM**ROTATE CUBE ABOUT Z-AXIS/SA
1810 FORI=1TONP
1820 X(I)=X(I)*COS(SA)-Y(I)*SIN(SA)
1830 Y(I)=Y(I)*COS(SA)+X(I)*SIN(SA)
1840 NEXT
1850 RETURN
```

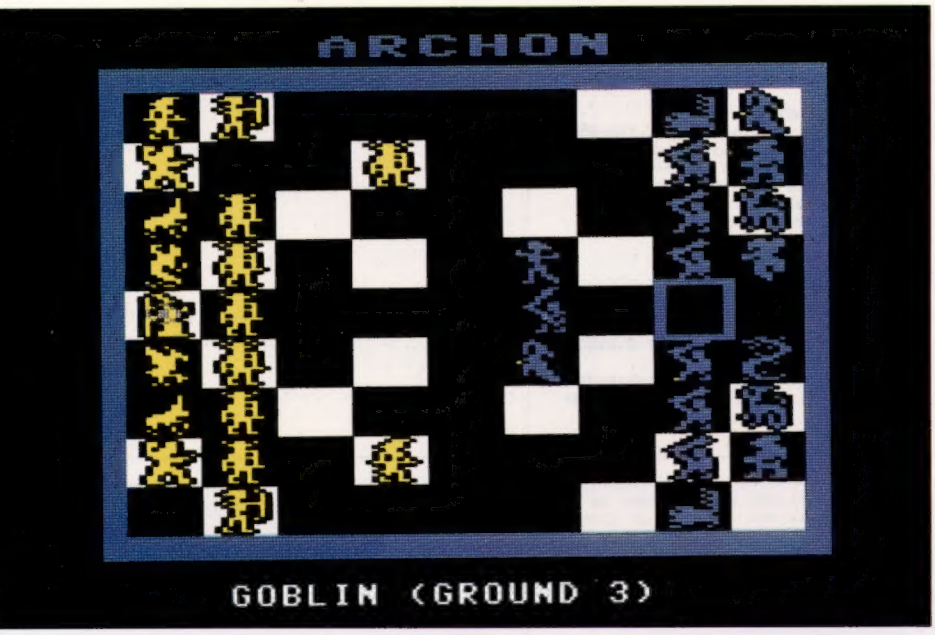
Programme Test I-ROTSUB

Le programme BASIC suivant doit être entré et sauvegardé comme TEST I-ROT. Notez qu'aucune variable ne doit être définie entre l'exécution du code de la ligne 1880 à 2000 et l'appel de SYS50523. Sinon, l'adresse de base des tableaux sera incorrectement passée au code machine et le programme se « plantera ».

```
1000 REM** TEST I-ROT **
1010 IFA=0THENA=1:LOAD"PLOTSUB.HEX",8,1
1020 IFA=1THENA=2:LOAD"LINESUB.HEX",8,1
1030 IFA=2THENA=3:LOAD"I-ROT.HEX",8,1
1040 REM**DIMENSION ARRAYS**
1050 NP=8:REM NUMBER OF POINTS
1060 DIM X(NP),Y(NP),Z(NP)
1070 DIM ED(NP,NP):REM EDGE CONNECTIONS
1080 REM**INITIALISE ARRAYS**
1090 REM--CUBE COORDINATE DATA
1100 DATA  75, 75, 75:REM----------/1
1110 DATA -75, 75, 75:REM TOP FOUR /2
1120 DATA -75,-75, 75:REM POINTS   /3
1130 DATA  75,-75, 75:REM----------/4
1140 DATA  75, 75,-75:REM----------/5
1150 DATA -75, 75,-75:REM BOT FOUR /6
1160 DATA -75,-75,-75:REM POINTS   /7
1170 DATA  75,-75,-75:REM----------/8
1180 REM**ROTATE SPACE ABOUT X-AXIS π/4
1190 FORI=1TONP
1200 READX(I),Y(I),Z(I)
1210 Y(I)=Y(I)*COS(π/4)-Z(I)*SIN(π/4)
1220 Z(I)=Z(I)*COS(π/4)+Y(I)*SIN(π/4)
1230 NEXT
1240 REM**ROTATE SPACE ABOUT Z-AXIS π/4
1250 FORI=1TONP
1260 X(I)=X(I)*COS(π/4)-Y(I)*SIN(π/4)
1270 Y(I)=Y(I)*COS(π/4)+X(I)*SIN(π/4)
1280 NEXT
1290 REM--EDGE CONNECTION DATA--
1300 E(1,2)=1:REM 1 CONNECTED TO 2
1310 E(2,3)=1:E(3,4)=1:E(4,1)=1
1320 E(5,6)=1:REM BOT SQUARE
1330 E(6,7)=1:E(7,8)=1:E(8,5)=5
1340 E(5,1)=1:REM TOP TO BOT EDGES
1350 E(6,2)=1:E(7,3)=1:E(8,4)=1
1360 REM**SYMMETRISE E(I,J)**
1370 FORI=1TONP:FORJ=1TONP
1380 IFE(I,J)=1THENE(J,I)=1
1390 NEXT:NEXT
1400 REM#########################
1410 REM##PLOT ROTATING CUBE##
1420 SA=2*π/45:CS=COS(SA):SN=SIN(SA)
1430 FOR A=0 TO 2*π STEP SA
1440 GOSUB1870:REM ROTATE THRU SA
1450 GOSUB1600:REM INIT/CLEAR SCREEN
1460 REM--PLOT CUBE--
1470 FORI=1TONP
1480 FORJ=1TOI
1490 IF E(I,J)=0THEN1520:REM NOT JOINED
1500 GOSUB1640:REM COMPUTE PROJECTION
1510 GOSUB1680:REM JOIN POINTS
1520 NEXT:NEXT
1530 REM-------------
1540 NEXT A:REM NEXT ANGLE
1550 REM#########################
1560 REM**WAIT**
1570 GETA$:IFA$=""THEN1570
1580 GOSUB1770:REM RESET SCREEN
1590 END
1600 REM**SETUP HIRES**
1610 POKE49408,1:POKE49409,1
1620 POKE49410,1:SYS49422
1630 RETURN
1640 REM**COMPUTE PROJECTION ON HIRES**
1650 X1%=X(I)+159:Y1%=199-(Z(I)+100)
1660 X2%=X(J)+159:Y2%=199-(Z(J)+100)
1670 RETURN
1680 REM**LINESUB**
1690 IF(X1%=X2%)AND(Y1%=Y2%)THENRETURN
1700 MHI=INT(X1%/256):MLO=X1%-256*MHI
1710 NHI=INT(X2%/256):NLO=X2%-256*NHI
1720 POKE49920,MLO:POKE49921,MHI
1730 POKE49922,NLO:POKE49923,NHI
1740 POKE49924,Y1%:POKE49925,Y2%
1750 SYS49934:REM LINESUB
1760 RETURN
1770 REM**RESET SCREEN**
1780 POKE49408,0:SYS49422
1790 PRINTCHR$(147)
1800 RETURN
1810 REM**ROTATE CUBE ABOUT Z-AXIS/SA
1820 FORI=1TONP
1830 X(I)=X(I)*CS-Y(I)*SN
1840 Y(I)=Y(I)*CS+X(I)*SN
1850 NEXT
1860 RETURN
1870 REM**ROTATE ABOUT Z-AXIS/SA
1880 X(1)=X(1):REM X(1) CURRENT VAR
1890 POKE50500,PEEK(71):REM X(1)LO
1900 POKE50501,PEEK(72):REM X(1)HI
1910 Y(1)=Y(1):REM Y(1) CURRENT VAR
1920 POKE50502,PEEK(71):REM Y(1)LO
1930 POKE50503,PEEK(72):REM Y(1)HI
1940 POKE50504,NP:REM NUMBER OF POINTS
1950 CS=CS:REM MAKE CS CURRENT VAR
1960 POKE50509,PEEK(71)
1970 POKE50510,PEEK(72)
1980 SN=SN:REM MAKE SN CURRENT VAR
1990 POKE50511,PEEK(71)
2000 POKE50512,PEEK(72)
2010 SYS50523
2020 RETURN
```

Forces élémentaires

En combinant certaines caractéristiques des échecs et des graphiques à action rapide, Archon, de Ariolasoft, met à l'épreuve l'habilité du joueur.



Un damier particulier
Archon peut être défini comme étant un jeu d'échecs animé. Lorsqu'une pièce est déplacée vers un autre carré sur le damier, elle marche, rampe ou vole selon sa nature. Dès qu'elle atteint le nouveau carré, s'il est occupé par une des pièces de l'opposant, l'écran affiche une vue rapprochée du carré et un combat s'engage. Le résultat du combat est fonction de l'habilité du joueur et des forces relatives des pièces opposées.
(Cl. Dimension Graphics.)

Archon est un jeu qui peut fasciner le joueur d'échecs tout autant que le plus grand amateur de jeu de café. Le jeu est basé sur une lutte opposant les forces de la « Lumière » aux forces de l'« Obscurité ». Le jeu commence par l'affichage d'un « damier » stratégique sur lequel les deux opposants sont positionnés en lignes et en colonnes, un peu comme sur un jeu d'échecs. Les pièces portent des noms comme « phoenix » et « knight » du côté de la lumière, et « banshees », « goblins » et « dragons », du côté de l'obscurité. Les icônes qui représentent les pièces des deux côtés ont diverses forces et possibilités de mouvement. Certaines pièces peuvent, par exemple, « voler », c'est-à-dire qu'elles peuvent sauter au-dessus des pièces qui se trouvent en face d'elles, tandis que d'autres ne peuvent effectuer que des mouvements au sol.

Le damier lui-même est divisé en une matrice neuf par neuf. A première vue, il ressemble à un damier ordinaire, avec ses carrés colorés alternativement en noir et en blanc. Cependant, d'autres carrés passent du noir au blanc puis redeviennent noirs lors du déroulement du jeu. Cela s'explique par le fait que les forces de la lumière sont plus fortes sur les carrés blancs et que les forces de l'obscurité sont plus fortes sur le noir.

L'objectif d'Arcon est d'occuper les cinq « points de puissance »; quatre sont positionnés en croix sur les bords du damier, le centre de la croix étant situé sur le carré du milieu. Lorsque le jeu commence, la stratégie initiale veut que vous déplaciez vos icônes depuis les couleurs opposées où ils sont vulnérables vers les couleurs où ils sont plus puissants. Ainsi, vous pouvez ouvrir la ligne du fond afin que les pièces « terrestres » soient libres de se déplacer pour construire une barrière de défense contre les forces terrestres de l'opposant.

A ce stade, le jeu ressemble fort à un jeu d'échecs sur ordinateur. Cependant, la réelle différence apparaît lorsqu'on tente d'occuper un carré déjà pris par une icône opposée. Lors du déplacement d'une icône vers un carré investi par un ennemi, l'écran affiche une vue rapprochée du carré au lieu de la prise de la pièce. Les pièces ennemies sont positionnées de chaque côté du carré, sur lequel on aperçoit une barre qui représente la force de l'icône; c'est à ce moment que le tir de jeu de café commence. Chaque fois que la barre est atteinte par la pièce opposée, la force diminue, et lorsqu'elle disparaît complètement, l'opposant occupe le carré. Puisque les forces et les méthodes d'attaque diffèrent radicalement d'une icône à l'autre, certains de ces combats sont plus inégaux que d'autres. Par exemple, un dragon peut cracher des flammes vers un opposant à travers le carré, tandis qu'un cavalier doit être très près de l'icône d'un ennemi pour utiliser son épée. Les combats sont encore compliqués par des barrières disposées sur l'écran qui changent de position pendant le déroulement du jeu.

Chaque côté a une icône qui peut remplir le rôle d'un ensorceleur : un sorcier pour la lumière, une sorcière pour l'obscurité! Les ensorceleurs sont identiques, mais ne peuvent être utilisés par l'icône qu'une seule fois. L'ensorceleur a le pouvoir de définir de nouvelles possibilités de mouvements pour une pièce particulière, ou de diminuer sa puissance initiale. Ce personnage peut servir à attaquer une pièce ennemie et est pratique lorsqu'une pièce ennemie attaque l'une de vos pièces beaucoup plus faible, ou si vous désirez affaiblir les pièces de votre opposant. Cependant, même si l'ensorceleur détruit l'icône ennemie, il disparaît à la fin du combat.

Archon : pour le Commodore 64.
Éditeur : Ariolasoft.
Auteurs : Anne Westfall, Jon Freeman, Paul Reiche III.
Manche à balai : nécessaire.
Format : cassette.

Page manquante
(publicité)

Page manquante
(publicité)